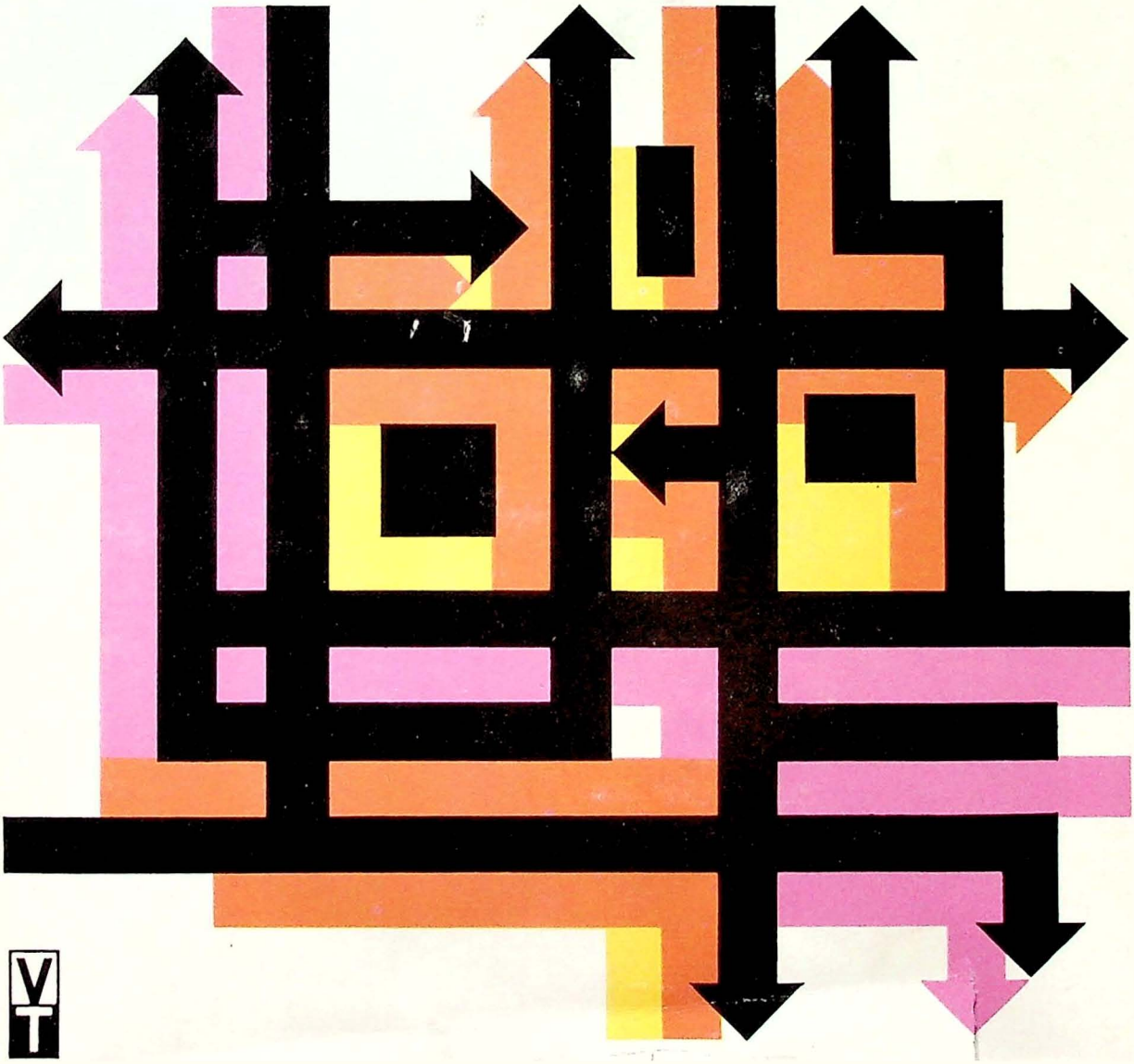
Matschke

Von der einfachen Logikschaltung zum Mikrorechner



VT

Mikroprozessoren und Mikrorechner bieten ständig neue Möglichkeiten in der funktionellen Gestaltung elektronischer Geräte und bei der automatischen Steuerung technischer Prozesse. Techniker aller Fachrichtungen, die sich bisher noch nicht mit den Problemen der digitalen Rechentechnik oder gar mit denen der Mikrorechentechnik befaßt haben, erkennen jetzt, daß zur effektiven Nutzung dieser neuen Technik in allen Bereichen der Volkswirtschaft, neben fundierten Kenntnissen über den zu steuernden Prozeß, auch Einblicke in die Funktionsweise mikroelektronischer Bauelemente gehören.

Besonders die heranwachsende junge Generation, die in den kommenden Jahren die neue Technik zu meistern hat, muß in verstärktem Maße an diese Thematik herangeführt werden.

Das Buch ist vor allem deshalb geschrieben worden, um all denen zu helfen, die sich möglichst rasch in die Grundlagen der digitalen Mikroelektronik, insbesondere die der Mikrorechner, einarbeiten wollen. Um dabei einen relativ breiten Leserkreis erfassen zu können, werden nur geringe Kenntnisse in Mathematik und Elektrotechnik vorausgesetzt.

In der vorliegenden Auflage sind u. a. die Wirkungsprinzipien neuer Mikroprozessorgenerationen und neuer Speicherbausteine aufgenommen worden.

VEB
Verlag Technik
Berlin

Von der einfachen Logikschaltung zum Mikrorechner

Dr.-Ing. Joachim Matschke

3., bearbeitete Auflage

Matschke, Joachim:
Von der einfachen Logikschaltung zum Mikrorechner/
Joachim Matschke. – 3., bearb. Aufl. – Berlin: Verl.
Technik, 1986. – 232 S.: 216 Bilder, 86 Taf.

ISBN 3-341-00069-0

3., bearbeitete Auflage
© VEB Verlag Technik, Berlin, 1986
Lizenz 201 · 370/69/86
Printed in the German Democratic Republic
Gesamtherstellung: Offizin Andersen Nexö, Graphischer Großbetrieb, Leipzig III/18/38
Lektor: *Karl Belter*
Schutzumschlag: *Kurt Beckert*
LSV 3053 · VT 3/5524-3
Bestellnummer: 553 606 0
02000

Vorwort

Die Mikroelektronik ist zu einem bestimmenden Wachstumsfaktor des wissenschaftlich-technischen Fortschritts geworden. Für den universellen Steuerbaustein *Mikrorechner* werden ständig neue Einsatzgebiete erschlossen.

Die große Bedeutung dieser Thematik widerspiegelt sich in einer Fülle von Publikationen. Dabei werden meist die Fachleute angesprochen, während ein nur kleiner Teil der Veröffentlichungen zur ersten Übersichtsinformation bestimmt ist. Für den möglichst raschen Übergang vom allgemeinen zum tieferen fachlichen Verständnis sind bisher kaum Bücher herausgegeben worden. Das vorliegende Buch ist deshalb für all diejenigen als Hilfe gedacht, die sich in die Grundlagen der digitalen Mikroelektronik, insbesondere die der Mikroprozessoren und Mikrorechner, einarbeiten wollen.

Damit ein möglichst breiter Interessentenkreis Nutzen aus der Thematik ziehen kann, werden zum erfolgreichen Studium dieses Buches nur geringe Kenntnisse in Mathematik und Elektrotechnik vorausgesetzt. Zur Erleichterung des Eindringens ist der mathematische Aufwand bei der Darstellung logischer Zusammenhänge so klein wie möglich gehalten. Aus diesem Grund wurde die Schaltalgebra nur in elementarer Form eingeführt, und Minimierungsverfahren sind absichtlich ausgelassen.

Um ein schnelles und effektives Eindringen in die gesamte Problematik zu ermöglichen, beschränken sich die Betrachtungen auf die wesentlichen Sachverhalte, während andere, auch wichtige Einzelheiten übergangen werden. So werden z. B. Darstellung und Verarbeitung von Zahlen im BCD-Code ausgelassen, werden Multiplikation und Division nur für ganzzahlige Potenzen der Zahl 2 erläutert.

Die Wirkungsweise der einzelnen Funktionsgruppen wird vorzugsweise auf dem Niveau von Logiksymbolen und Signalflußbildern erklärt. Auf Schaltungsdetails, die die Einbeziehung von Spannungs- und Leistungspegelverhältnissen notwendig machen, wurde nur in wenigen Fällen eingegangen. Es wurden bewußt einfache, die prinzipiellen Aussagen unterstützende Beispiele (besonders bei den höherentwickelten Logikschaltungen) ausgewählt.

In der 3., bearbeiteten Auflage sind insbesondere die Wirkprinzipien neuer Mikroprozessorgenerationen und ihre Zusatzschaltkreise aufgenommen worden. Stark erweitert wurde der Abschnitt über Speicherbausteine, und es wurde eine Arithmetik-Logik-Einheit in leicht faßlicher Form dargestellt. Insgesamt wurde dem Weiterentwicklungstrend Rechnung getragen.

Für sachkundige Hinweise und Anregungen möchte ich Herrn Prof. Dr. sc. techn. *W. Schwarz* (Dresden), Herrn Dr.-Ing. *L. Richter* (Berlin) und Herrn *K.-H. Rumpf* (Berlin) herzlich danken. Mein besonderer Dank gilt dem Lektor, Herrn *K. Belter*, für die gute Zusammenarbeit. Weiterhin danke ich Herrn *G. Reimann* für das sorgfältige Bearbeiten der Zeichnungsvorlagen und Herrn *K. Hoppe*, der die ausgewogene typografische Gestaltung besorgte.

Berlin *Joachim Matschke*

Inhaltsverzeichnis

1. Einführung in die Informationsverarbeitung 9

1.1. Information 9

1.2. Formen der Informationsdarstellung 10
1.2.1. Bildzeichensysteme 10
1.2.2. Buchstabensysteme 11
1.2.3. Zahlensysteme 12
1.2.4. Zustandssysteme 17

1.3. Transformation der Informationsdarstellung von einem Zeichensystem in das andere 19

1.4. Informationsverarbeitung 23
1.4.1. Programmablaufplan 23
1.4.2. Addition und Subtraktion 26
1.4.3. Automatische Zustandserkennung mit Folgereaktion 31
1.4.4. Informationsaustausch zwischen einzelnen Teilsystemen 32
1.4.5. Zusammenwirken von mehreren informationsverarbeitenden Systemen 37

2. Informationsverarbeitung durch einfache Logiksysteme 39

2.1. Einfache elektrische Logiksysteme ohne Rückführung 39
2.1.1. Elementare Logikschaltungen 39
2.1.1.1. UND-Funktion 39
2.1.1.2. ODER-Funktion 42
2.1.1.3. Negation 43
2.1.1.4. NAND-Funktion 44
2.1.1.5. NOR-Funktion 45
2.1.2. Theoreme von *de Morgan* 45
2.1.3. Regeln und Anwendung der Schaltalgebra 46
2.1.4. Einfluß der Halbleitertechnik 50
2.1.5. Der Transistor als Schalter 51
2.1.5.1. Bipolartransistor 52
2.1.5.2. Unipolartransistor 54
2.1.6. Übergang zu allgemeinen Logikschaltungssymbolen 55
2.1.7. Wichtige Grundschaltungen 57
2.1.7.1. Auswahl eines Logikzustands auf einer Leitung 58
2.1.7.2. Signaleinschalter (Torschaltungen) 59
2.1.7.3. Signalumschalter (Multiplexer, Demultiplexer) 60
2.1.7.4. Sammelschienenübertragung (Bus) 63
2.1.7.5. Vergleicher 65
2.1.7.6. Zuordner (Codierung, Umcodierung, Decodierung) 66
2.1.7.7. Festwertspeicher: ROM 72
2.1.7.8. Programmierbare Logikanordnung 80
2.1.7.9. Schaltungen zur Addition und Subtraktion 83
2.1.7.10. Arithmetik-Logik-Einheit (ALU) 85

2.2. Einfache elektrische Logiksysteme mit Rückführung 88
2.2.1. Elementare Logikschaltungen mit Rückführung 89
2.2.1.1. Monostabiler Multivibrator (Impulsdehner) 89
2.2.1.2. Astabiler Multivibrator (Pulsgenerator) 91
2.2.1.3. RS-Trigger aus NAND-Gliedern 92

2.2.1.4. RS-Trigger aus NAND-Gliedern mit vorgeschalteten Negatoren 95
2.2.1.5. Getakteter RS-Trigger 96
2.2.1.6. D-Trigger 97
2.2.1.7. D-Trigger mit Taktflankensteuerung 97
2.2.1.8. JK-Master-Slave-Trigger 100
2.2.2. Zusammenschaltung von Triggern 102
2.2.3. Register 104
2.2.3.1. Parallelregister 105
2.2.3.2. Schieberegister für serielle Eingabe 105
2.2.3.3. Schieberegister für serielle oder parallele Eingabe 108
2.2.3.4. Weitere Anwendungsbeispiele für Schieberegister 109
2.2.3.5. Andere Ausführungsformen von Schieberegistern 110
2.2.4. Binärzähler 112
2.2.4.1. Asynchrone Binärzähler 112
2.2.4.2. Synchrone Binärzähler 117
2.2.4.3. Universeller Zählerbaustein 120
2.2.4.4. Einige Anwendungen des Zählers 120
2.2.5. Speicherschaltungen vom RAM-Typ 123
2.2.5.1. Bipolare Speicherzelle 123
2.2.5.2. MOS-Speicherzelle 124
2.2.5.3. Aufbau von RAM 126

3. Informationsverarbeitung durch höher entwickelte Logiksysteme 128

3.1. Bitmustergenerator 129

3.2. Programmierbarer Funktionsgenerator 132

3.3. Mustergenerator 135

3.4. Verbesserungsmöglichkeiten des Mustergenerators 141

3.5. Matrixsteuerwerk 143

3.6. Auf dem Wege zum universellen Steuersystem 150

3.7. Mikroprozessor 155

3.8. Einfacher Mikroprozessor 158
3.8.1. Befehlsliste 159
3.8.2. Entwurfsüberlegungen zu Einzelheiten des Blockbilds 165
3.8.3. Entwurfsüberlegungen zum Mikrosteuerwerk 165
3.8.4. Die ersten kleinen Programme 173
3.8.5. Programmierbeispiel 177

3.9. Programmherstellung 184
3.9.1. Programmherstellung mit Hilfe der Assemblersprache 186
3.9.2. Unterprogrammtechnik 189
3.9.3. Definition von Makrobefehlsfolgen 192

3.10. Weiterentwicklung des Mikroprozessor/Mikrorechner-Konzepts 194
3.10.1. Bussystem 195
3.10.2. Arithmetik-Logik-Einheit 196
3.10.3. Registerblock 196
3.10.4. Speicherverwaltungseinheit/Speicherblock 201
3.10.5. Mikrosteuerwerk 204
3.10.6. Zähler/Zeitgeber-Block (CTC) 205
3.10.7. Eingabe-Ausgabe-Einheit (E/A-Block) 209

3.10.7.1. Eingabe-Ausgabe-Einheit für serielle Datenübertragung 209
3.10.7.2. Eingabe-Ausgabe-Einheit für parallele Datenübertragung 212
3.10.7.3. Steuerung der Eingabe-Ausgabe-Prozesse 213
3.10.8. Gestaltung des Unterbrechungssystems 214

3.11. Anwendungsmöglichkeiten des Mikrorechners 220

4. Erläuterung spezieller Begriffe 226

Literaturverzeichnis 229

Sachwörterverzeichnis 231

1. Einführung in die Informationsverarbeitung

1.1. Information

Der Begriff *Information* stammt aus dem Lateinischen und hat im allgemeinen Sprachgebrauch die Bedeutung von Auskunft, Benachrichtigung, Nachricht, Mitteilung, Belehrung. Er bezeichnet den Vorgang der Übermittlung von Neuigkeiten, aber auch den auswertbaren Inhalt dieser Neuigkeit. Ein Beispiel soll diese Aussagen veranschaulichen. Dazu wollen wir uns eine Drahtverbindung vorstellen, über die mit Hilfe der Morsetelegrafie der Text gesendet werden soll: „Habe keine Zeit." Dann sieht die durch Betätigen der Morsetaste entstehende Stromstärkeschwankung wie im **Bild 1.1** dargestellt aus. Der Sendetext ist eine Folge von kurzen und langen Stromimpulsen mit entsprechenden Pausen umgewandelt worden. Der Empfänger kann diese Stromimpulsfolge mit Hilfe des Kopfhörers abhören und wieder in den Klartext zurückübersetzen.

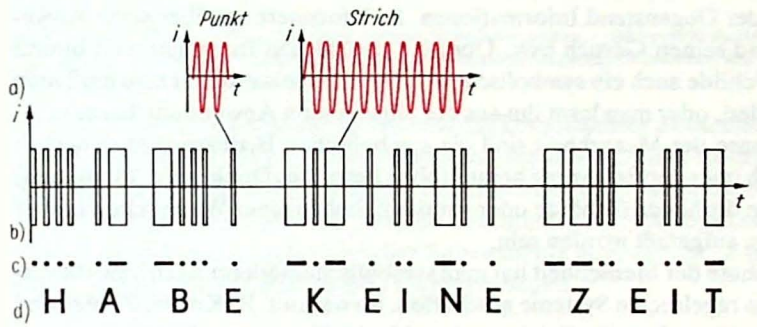


Bild 1.1
Übertragung von Text durch Morsezeichen
a) zeitlicher Verlauf der Wechselstromimpulse für Punkt und Strich; b) zeitliche Aufeinanderfolge der Impulse; c)Text in Morseschrift; d) Klartext

Der gesamte Vorgang läßt sich in einzelne Bearbeitungsschritte zerlegen:

- Formulierung des Textes
- Umsetzung des Textes in Stromimpulse nach dem Morsealphabet (Verschlüsselung oder Codierung genannt)
- Abhören der Stromimpulsfolge und Rückübersetzen in den Klartext (Entschlüsselung oder *Decodierung*).

Wo steckt nun in diesem Beispiel die Nachricht oder Information? Sie ist offensichtlich im Text „Habe keine Zeit" enthalten, wobei dieser Text sowohl in der Schriftform als auch in Form von Punkten, Strichen und Zwischenräumen wie auch in Form einer Stromimpulsfolge oder deren Schallschwingung auftreten kann. Bei all diesen verschiedenen Zustandsformen – Schrift, elektrische Stromimpulsfolge, Schallschwingung – muß demnach der Anteil, der die Nachricht enthielt, unverändert geblieben sein. Geändert hat sich nur jeweils der Transportbehälter der Nachricht, der *materielle Träger*.
Man kann also feststellen, daß jede Nachricht einen materiellen Träger benötigt. Dieser materielle Träger kann die unterschiedlichsten materiellen Zustandsformen haben. Damit er aber eine Nachricht oder Information übertragen kann, muß er unterscheidbare Merkmale aufweisen, z. B. die un-

terschiedliche Linienführung bei der Schrift oder die unterschiedliche Länge der Impulse und der Pausen oder unterschiedliche Intensitätsgrade einer physikalischen Größe.
Verallgemeinernd kann man davon sprechen, daß der materielle Träger der Information ein bestimmtes *Muster* aufweisen muß. Durch dieses Muster kann man die Information darstellen. Einen gemusterten materiellen Träger, der zur Informationsübertragung verwendet wird, nennt man ein *Signal*. Welche Bedeutung ein entsprechendes Muster aufweist, kann je nach der Verabredung verschieden sein.

1.2. Formen der Informationsdarstellung

Wir wissen: Ein gemusterter materieller Träger, der eine Information übermittelt, wird als Signal bezeichnet.

Bedeutet das nun, daß man alle materiellen Gebilde unserer Umgebung als Signale auffassen kann?
Tatsächlich übermittelt uns jeder Gegenstand Informationen. Er informiert uns über seine Anwesenheit, seine Farbe, Form und seinen Geruch usw. Über die natürlichen Informationen hinaus kann nun jedem materiellen Gebilde auch ein symbolischer Sinn verliehen werden. Dazu muß man diesen Sinn entweder verabreden, oder man lernt ihn aus der allgemeinen Anwendung heraus.
In der frühen Entwicklungsphase der Menschheit sind die symbolischen Bedeutungen sicherlich aus dem allgemeinen Gebrauch oder der Erfahrung heraus (ohne bewußtes Denken des Menschen) entstanden. So dürfte z. B. eine drohende Gebärde oder gar das Erheben einer Waffe schon immer als Zeichen einer Kampfansage aufgefaßt worden sein.
In der späteren Entwicklungsphase der Menschheit hat man symbolische Bedeutungen bewußt verabredet und noch später daraus regelrechte Systeme geschaffen. So waren z. B. Krone, Zepter und entsprechende Gewänder eines Herrschers die Zeichen seiner Macht. Darum wurde dafür auch das lateinische Wort Insignien verwendet, das auf deutsch Zeichen, Anzeichen, im betrachteten Fall Kennzeichen der staatlichen Macht bedeutete.
Auch Laut- und Schriftsprache bilden ein *Symbolsystem*. Interessant ist dabei, daß die ersten Schriftsprachen aus Bildzeichen bestanden, also die Herleitung der Symbole aus den materiellen Gegenständen beweisen. Erst die fortlaufende Entwicklung der Symbolsysteme führte zur heute üblichen abstrakten Buchstabenschrift.

1.2.1. Bildzeichensysteme

Die Frühformen der künstlichen Symbole bestehen aus vereinfachten (abstrakten) Abbildungen natürlicher Gegenstände.

Hierzu gehören z. B. die altägyptischen Bilderschriftzeichen, die im 4. Jahrtausend vor unserer Zeitrechnung entstanden sind und erst im 3. Jahrhundert unserer Zeitrechnung von griechischen Schriftzeichen abgelöst wurden. Dazu gehört die chinesische Schrift, die über etwa 40000 Schriftzeichen verfügt, die aus Bildern hervorgingen.
Der Vorteil von Bildzeichen besteht in ihrer schnellen Lesbarkeit. Ihr Nachteil ist die kompliziertere Herstellungsweise. Dieses schnellere Erfassen von Bildzeichen hängt damit zusammen, daß

das menschliche Bildempfangssystem (Auge, Gehirn) sich im Verlauf der biologischen Entwicklung zu einem Informationsauswertungssystem höchster Qualität bezüglich Verarbeitungsgeschwindigkeit und Erkennung von Unterschieden ausgebildet hat. Daher werden Bildzeichen in all den Fällen angewendet, in denen rasches Erfassen von Informationen bedeutungsvoll ist.
In dieser Hinsicht müssen vor allem die *Verkehrszeichen* erwähnt werden, da hier mitunter nur Sekundenbruchteile für die Betrachtung zur Verfügung stehen.

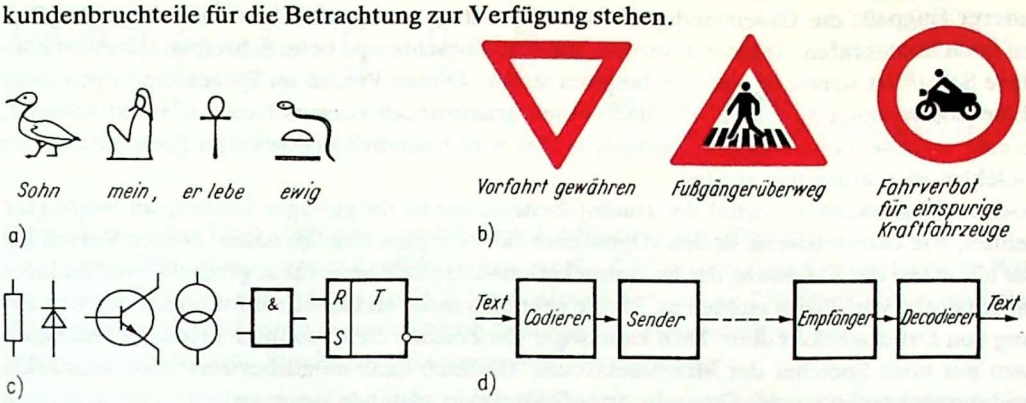


Bild 1.2. Bildzeichen

a) altägyptische Bilderschriftzeichen (Hieroglyphen); b) Verkehrszeichen; c) Bildsymbole elektronischer Bauelemente (von links nach rechts: Widerstand, Diode, Transistor, Konstantstromquelle, UND-Gatter, RS-Trigger); d) Blockbild einer Nachrichtenübertragungsstrecke

Aber auch zur übersichtlicheren Darstellung komplizierter Zusammenhänge bei technischen Systemen werden bevorzugt *Bildzeichensysteme* verwendet. Hier ist die Vielfalt der Bildsymbole inzwischen so groß geworden, daß die Beherrschung ohne fachliche Kenntnisse nicht mehr möglich ist. Beispielsweise werden in diesem Buch Symbole von elektronischen Systemen in Blockform und in mehr bildlicher Form für einzelne Bauelemente angewendet. Auf ihre Definition wird noch an entsprechender Stelle und im **Bild 1.2** eingegangen.

1.2.2. Buchstabensysteme

Die modernen Sprachen verwenden Schriftsysteme, die als kleinstes Element den Buchstaben aufweisen.
Aus Buchstaben werden Silben, aus Silben Wörter und aus Wörtern Sätze gebildet.

Der Sinn dieser Gliederung in Buchstaben, Silben, Wörter, Sätze liegt darin, die Schriftsprache der Lautsprache anzupassen und durch diesen Zusammenhang die Erlernbarkeit zu erleichtern. Bleibt noch zu erläutern, warum sich die akustischen Zeichen der lautsprachlichen Äußerung vom Ursprung her so stark von den bildlichen Zeichen der Schriftsprache unterscheiden. Das liegt im Aufbau der Sende- und Empfangssysteme und in den Eigenschaften der Übertragungsmedien begründet.
Sprachliche Laute werden zeitlich nacheinander erzeugt, durch Schallwellen übertragen und vom Ohr empfangen. Unser akustisches Empfangssystem kann dabei eine Entschlüsselung der Nachricht nur dann ohne Schwierigkeiten realisieren, wenn nur eine Sprachschwingung einwirkt.
Anders ist es beim Bildempfang. Hier können viele Informationen gleichzeitig aufgenommen und ausgewertet werden, da unsere Netzhaut für den gleichzeitigen Empfang von vielen Bildelementen ausgerüstet ist. Mann nennt diese Art *parallele Informationsübertragung*, während die lautsprachliche Verständigung ein Beispiel für *serielle Informationsübertragung* darstellt. Wir werden sehen,

daß diese beiden Übertragungsarten grundlegende Bedeutung haben. Die Verarbeitung parallel übertragener Informationen ist mit höherem Aufwand verbunden, erfolgt aber entsprechend schneller und zuverlässiger als die langsamere serielle Informationsverarbeitung.
Demzufolge ist die Anpassung der Schriftsprache an die Lautsprache mit einem Verlust an möglicher *Übertragungsgeschwindigkeit* (Lesegeschwindigkeit) verbunden. Das stimmt. Aber da ist noch ein anderer Engpaß: die Geschwindigkeit beim Schreiben der sprachlichen Äußerung. Zur Zeit müssen noch Stenografen eingesetzt werden, um dem Sprachtempo beim Schreiben folgen zu können. Ihre Schrift ist schwierig und nur langsam lesbar. Diesen Prozeß im Sprachtempo von einer Maschine übernehmen zu lassen, die dann einen grammatisch einwandfreien Klartext schreibt, würde eine enorme Zeiteinsparung bedeuten. Das wird sicherlich bei weiteren Fortschritten der Mikroelektronik realisierbar werden.
Ein noch nicht genannter Vorteil der Buchstabensysteme ist ihr geringer Umfang an benötigten Elementen, wie beispielsweise in den Alphabeten der europäischen Sprachen. Dieser Vorteil hat sich vor allem seit der Erfindung der Buchdruckerkunst deutlich bemerkbar gemacht, weil dadurch die Lagerhaltung von *Typen* einfach ist. Heute geht man mehr und mehr zur fotomechanischen Erzeugung von Druckstöcken über. Man kann sogar die Zeichen elektrooptisch erzeugen und benötigt dazu nur noch Speicher der Mikroelektronik. Dadurch kann möglicherweise eine verstärkte Anwendung mehr oder weniger komplizierter Bildzeichen zustande kommen.

1.2.3. Zahlensysteme

Die Einführung des Zahlbegriffs ist eine der größten menschlichen Abstraktionsleistungen. Es mußten Begriffe erfunden werden, die als konkrete Gegenstände nicht vorlagen.

Wahrscheinlich entstand dies aus der Notwendigkeit, für Anhäufungen irgendwelcher Objekte Vergleichsangaben zu ermitteln. So wurde z. B. bei der Entrichtung von Naturalabgaben die jeweilige Menge erfaßt, indem man je Mengeneinheit einen Zählgegenstand (Stein oder Astholz) in einen Behälter (Speicher) legte. Bei der Mengenfeststellung der Zählholze mußten dann die Wörter aufgezählt werden, die die Folge der natürlichen Zahlen darstellen. Durch die häufige Wiederholung dieser Wortfolge hat sich diese bei uns fest eingeprägt. Hinzu kommen noch die *Bildungsgesetze* für unser *Zehnersystem*, so daß wir beliebig lange Zahlenfolgen aufzuzählen gelernt haben.
Bei der schriftlichen Formulierung der Zahlzeichen gab es eine ähnliche Entwicklung von der Bilderschrift zur Schrift mit Systemelementen wie bei der *Notierung* der übrigen Lautsprache. Allerdings endete die Entwicklung in einer anderen Systematik, da die Zahlen einem Informationsverarbeitungsverfahren unterzogen werden, das bei den anderen lautsprachlichen Äußerungen nicht angewendet wird. Gemeint sind die *Rechenoperationen*. Der effektiven Ausführung dieser Rechenoperationen ordnen sich alle anderen Überlegungen unter. So gab z. B. die Erhöhung der Rechengeschwindigkeit den Ausschlag dafür, daß die arabischen Ziffern die römischen Zahlzeichen ablösten.
Auch in der heutigen Anwendung der Zahlensysteme wählt man das System nach seinen Eigenschaften aus. So arbeiten wir Menschen, weil es seit Generationen so geübt worden ist, mit dem *Dezimalsystem*. Daß auch andere Systeme im Gebrauch waren, beweist das *Sexagesimalsystem* (Sechzigersystem) der Babylonier, nach dem wir noch heute die Stunde in 60 Minuten und die Minute in 60 Sekunden einteilen.
In Rechenautomaten wird dagegen das *Binärsystem* (Zweiersystem, z. B. Dualzahlensystem) benutzt, das aus nur zwei Zahlzeichen besteht und in der elektronischen Realisierung die höchste Zuverlässigkeit gegenüber Störeinflüssen aufweist.

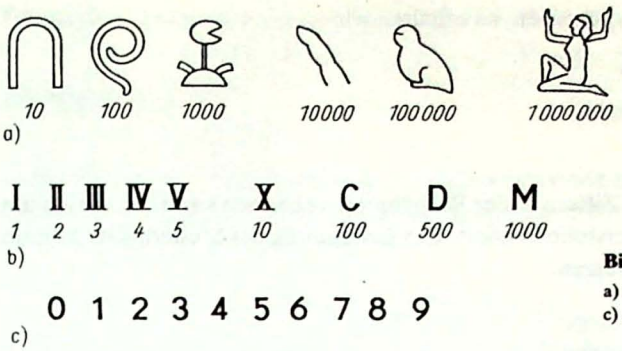


Bild 1.3. Zahlzeichen
a) altägyptische Zahlzeichen; b) römische Zahlzeichen; c) arabische Ziffern

Bild 1.3 zeigt eine Übersicht über verschiedene Zahlzeichendarstellungen.
Entwickeln wir uns nun einmal das Bildungsgesetz für Zahlen nach dem Dezimalsystem. Wir wählen uns als Beispiel, eine Menge von 125 Stück abzuzählen. Diese 125 Stück sollen zunächst, wie **Bild 1.4** zeigt, durch Striche in einer Strichliste symbolisch dargestellt werden.
Wir schreiben dazu die Striche nach folgender Anordnung nieder:

- Nach je vier Strichen wird der fünfte Strich als Querstrich durch die vorangegangenen vier Striche gezogen.
- Je zwei solcher Fünfergruppen werden nebeneinander angeordnet und bilden eine Zehnergruppe.
- Je fünf Zehnergruppen werden untereinander geschrieben und ergeben eine Fünfzigergruppe.
- Zwei Fünfzigergruppen werden jeweils nebeneinander angeordnet und bilden eine Hundertergruppe.

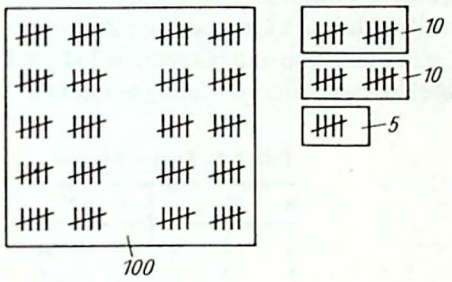


Bild 1.4
Ordnung von Zählstrichen nach dem Dezimalsystem am Beispiel der Abzählung von 125 Stück

Die Auswertung wird dann so durchgeführt, daß zunächst die Anzahl der Gruppen mit dem höchsten Wert ermittelt wird. Im Beispiel ist es die Gruppe mit der Wertigkeit hundert (Hundertergruppe), die einmal vorhanden ist. Der verbleibende Rest wird auf die Anzahl der Gruppen mit der nächst niederen Wertigkeit untersucht. Man findet zweimal die Zehnergruppe. Der noch verbleibende Rest enthält noch fünf Einer.
Dann notieren wir das Ergebnis der Zählung:

$$1 \times 100$$
$$2 \times 10$$
$$5 \times 1$$

oder in anderer Form, wenn wir mit x das Ergebnis der Zählung bezeichnen:

$$x = 1 \cdot 100 + 2 \cdot 10 + 5 \cdot 1.$$

Benutzen wir die Schreibweise mit Zehnerpotenzen, so erhalten wir

$$x = 1 \cdot 10^2 + 2 \cdot 10^1 + 5 \cdot 10^0.$$

Aus dieser Darstellung läßt sich die Kurzform

$$x = 125$$

ableiten, bei der man aus der Stellung der Ziffern in der Reihenfolge erkennen kann, ob es sich um die Einer-, die Zehner- oder die Hunderterstelle handelt. Zur Erkennung des Stellenwerts könnte man auch das *Kästchenchema* **Bild 1.5** benutzen.

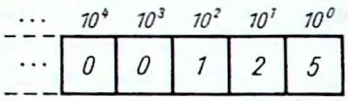


Bild 1.5
Stellenwertschema für Dezimalzahlen

Das läßt sich für beliebige Zahlen entsprechend fortsetzen. So ist z. B.

$$x = 2864 = 2 \cdot 10^3 + 8 \cdot 10^2 + 6 \cdot 10^1 + 4 \cdot 10^0.$$

Allgemein sind unsere Dezimalzahlen also mit folgender Summe darstellbar:

$$x = a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \ldots + a^1 \cdot 10^1 + a_0 \cdot 10^0.$$

Daraus läßt sich mit Hilfe des Summenzeichens $\sum$ und der Zählgröße i (i geht von Null bis n) folgende Kurzform schreiben:

$$x = \sum_{i=0}^{n} a_i \cdot 10^i.$$

Drückt man den eben im Zehnersystem beschriebenen Sachverhalt im *Dualsystem* aus, in dem nur die beiden Ziffern 0 und 1 definiert sind, so muß man die Zählstriche nach Gruppen von Zweierpotenzen ordnen. Die Potenzen der Basis zwei sind für die Exponenten von null bis zehn in **Tafel 1.1** zusammengestellt. In unserem Beispiel muß man die im **Bild 1.6** abgebildete Ordnung herstellen.

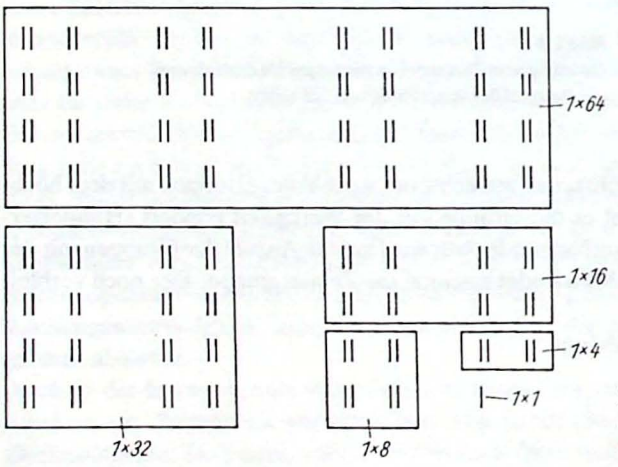


Bild 1.6
Ordnung von Zählstrichen nach dem Dualsystem am Beispiel der Abzählung von 125 Stück

Tafel 1.1. Zweierpotenzen

n	2^n	n	2^n
0	1	6	64
1	2	7	128
2	4	8	256
3	8	9	512
4	16	10	1024
5	32		

Dann beginnt man mit der Ermittlung desjenigen Anteils, der die größte Zweierpotenz darstellt, und erhält $1 \cdot 64 = 1 \cdot 2^6$. Danach erfolgt die Überprüfung, ob die nächstkleinere Zweierpotenz $2^5 = 32$ im Rest enthalten ist. Ergebnis: $1 \cdot 32$ ist vorhanden. In diesem Sinn verfährt man weiter bis zur 2^0.

Die nach Zweierpotenzen geordnete Summe erhält man zu

$$x = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0.$$

Die Kurzform lautet

$$x = 1111101$$

oder läßt sich mit Kennzeichnung der Stellenwerte wie im **Bild 1.7** darstellen. Die Anzahl der verwendeten Stellen wird in Bit (binary digit, Binärschritt) angegeben. So hat das im Bild 1.7 verwendete Format die Zahl von 8 Bit.

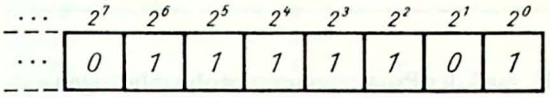


Bild 1.7
Stellenwertschema für Dualzahlen

Um auch gebrochene Zahlen darstellen zu können, setzt man die Reihenentwicklung zu *negativen Exponenten* hin fort. So wird z. B. die Dezimalzahl $x = 12{,}678$ wie folgt gewertet:

$$x = 1 \cdot 10^1 + 2 \cdot 10^0 + 6 \cdot 10^{-1} + 7 \cdot 10^{-2} + 8 \cdot 10^{-3}.$$

Die Stellen hinter dem Komma werden also durch Zehnerpotenzen mit negativen Exponenten bewertet.
In gleicher Weise geht man bei Dualzahlen vor. So entstpricht die Zahl $x = 101{,}011$ folgender Summendarstellung:

$$x = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3}.$$

Umgerechnet entsteht daraus

$$x = 4 + 1 + \tfrac{1}{4} + \tfrac{1}{8}$$

$$x = 5{,}375.$$

Am Beispiel einer festen Wortbreite von 8 Bit = 1 Byte (s. S. 33 und Abschn. 4.) wollen wir uns die Wirkung der Kommastelle näher betrachten.
Dazu sind im **Bild 1.8** die Wertigkeitsverteilungen für eine echt gebrochene und eine unecht gebrochene Dualzahl dargestellt.

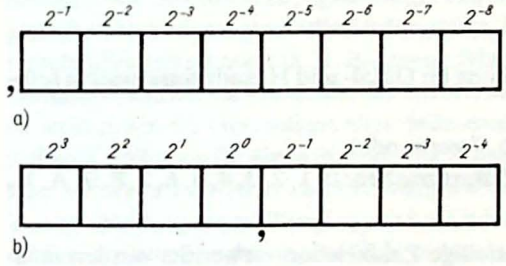


Bild 1.8
Stellenwertschema für gebrochene Dualzahlen
a) echt gebrochene Dualzahl; b) unecht gebrochene Dualzahl

Es ist interessant, daß eine Verschiebung des Kommas nach rechts eine *Multiplikation* mit einer Zweierpotenz bedeutet. Eine Kommaverschiebung nach links entspricht dagegen einer *Division* durch eine Zweierpotenz.

Beispiel:

$$\begin{aligned} x = 0100{,}1000 &= 4{,}5 \\ 01001{,}000 &= 9 \quad = 2 \cdot x \\ 010010{,}00 &= 18 \quad = 4 \cdot x \\ 010{,}01000 &= 2{,}25 \quad = x/2 \\ 01{,}001000 &= 1{,}125 = x/4. \end{aligned}$$

In **Tafel 1.2** finden Sie dazu eine Zusammenstellung.

Tafel 1.2. Auswirkung der Kommaverschiebung bei Dualzahlen

Verschiebung des Kommas um n Stellen	nach rechts entspricht einer Multiplikation mit	nach links entspricht einer Division durch
1		2
2		4
3		8
4		16
5		32
usw.		usw.

Ob ein Komma definiert wird und an welche Stelle, muß der Programmierer problembezogen entscheiden. Die *Gleitkommadarstellung*, die Sie vom Taschenrechner her kennen, soll hier nicht besprochen werden. Ebenso soll hier nicht die Darstellung im BCD- Code, die direkte Codierung jeder Zehnerstelle als Dualzahl, behandelt werden.
Im Prinzip kann man Zahlensysteme mit beliebigen Basiszahlen definieren. Im Gebrauch sind noch das Oktalsystem (Basis 8) und das Hexadezimalsystem (Basis 16).
Um nicht bei diesen vielen Systemen durcheinanderzugeraten, muß man die Zahlen bei gleichzeitiger Benutzung verschiedener Systeme entsprechend kennzeichnen. Eine Möglichkeit besteht darin, einen Index an den Variablen anzubringen, z. B. x_2, x_8, x_{16}. Eine andere Möglichkeit ist durch Anfügen eines Buchstabens an die Zahl gegeben: 0101 B (Binärzahl), 72 Q (Oktalzahl), 80 H (Hexadezimalzahl). **Tafel 1.3** zeigt eine Gegenüberstellung dieser Möglichkeiten.

Zahlensystem	Basis	Kennzeichnung A	Kennzeichnung B
Dualsystem	2	x_2	0101 B
Oktalsystem	8	x_8	72 Q
Dezimalsystem*)	10	x	126
Hexadezimalsystem	16	x_{16}	80 H

*) Die Dezimalzahl erhält keine Kennzeichnung.

Tafel 1.3
Kennzeichnung der Zahlen verschiedener Systeme

Zum Abschluß wollen wir uns noch die Zahlendarstellung im Oktal- und Hexadezimalsystem selbständig erarbeiten. Dazu noch folgende Hinweise:

- Im Oktalsystem werden die Ziffern 0, 1, 2, 3, 4, 5, 6, 7 verwendet.
- Das Hexadezimalsystem besteht aus folgenden 16 Ziffernzeichen: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Die Buchstaben wurden eingesetzt, damit nicht zweistellige Zahlzeichen verwendet werden müssen, die dann zu Irrtümern Anlaß geben. So stehen die Buchstaben für folgende Zahlen:

A = 10
B = 11
C = 12
D = 13
E = 14
F = 15.

Versuchen Sie nun bitte einmal selbst, ausgehend von der Stückzahldarstellung durch Striche und der entsprechenden Zusammenfassung in Gruppen, *die Zahl 125 im Oktal- und im Hexadezimalsystem darzustellen.* (Ergebnis: x = 175 Q = 7D H.)

In **Tafel 1.4** finden Sie eine Gegenüberstellung der Zahlen von 0 bis 32 im Dual-, Oktal-, Dezimal- und Hexadezimalsystem.

Tafel 1.4. Darstellung der Zahlen von 0 bis 32 in verschiedenen Zahlensystemen

Dual	Oktal	Dezimal	Hexadezimal	Dual	Oktal	Dezimal	Hexadezimal
000000	0	0	0	010001	21	17	11
000001	1	1	1	010010	22	18	12
000010	2	2	2	010011	23	19	13
000011	3	3	3	010100	24	20	14
000100	4	4	4	010101	25	21	15
000101	5	5	5	010110	26	22	16
000110	6	6	6	010111	27	23	17
000111	7	7	7	011000	30	24	18
001000	10	8	8	011001	31	25	19
001001	11	9	9	011010	32	26	1A
001010	12	10	A	011011	33	27	1B
001011	13	11	B	011100	34	28	1C
001100	14	12	C	011101	35	29	1D
001101	15	13	D	011110	36	30	1E
001110	16	14	E	011111	37	31	1F
001111	17	15	F	100000	40	32	20
010000	20	16	10				

1.2.4. Zustandssysteme

Unter einem Zustand versteht man die Gesamtheit der erkennbaren Eigenschaften eines materiellen Gebildes oder eines materiellen Prozesses zu einem bestimmten Zeitpunkt.

Erkennbare Eigenschaften können über unsere Sinnesorgane wahrgenommen werden, z. B. Farbe, Form, Temperatur eines Gegenstands. Sie können aber auch mit Hilfe von Meßgeräten festgestellt werden, wie Stromstärke, Spannung, Frequenz einer elektrischen Schwingung. Die einzelnen Eigenschaften können graduelle Stufungen (z. B. 1A, 2A, 3A usw.) oder auch sonstwie geartete Unterscheidungsmerkmale (z. B. Fahrzeug fährt, Fahrzeug steht still) aufweisen.

Definiert man solche Unterschiede derart, daß eindeutig abgrenzbare Bereiche festgelegt werden, so wollen wir die Gesamtheit aller definierten Bereiche für bestimmte Eigenschaften eines materiellen Gebildes oder eines materiellen Prozesses ein *Zustandssystem* nennen. Beispiele für solche Zustandssysteme sind in **Tafel 1.5** dargestellt.

Zur Herstellung einer Rangfolge oder Ordnung in einem Zustandssystem ist es zweckmäßig, die einzelnen Bereiche durch eine Zahl reihenfolgemäßig festzulegen. Man kommt dann zu einem geordneten Zustandssystem. Im Fall gradueller Stufungen wird man die *Ordnungszahlen* vernünftigerweise so zuordnen, daß die nächstgrößere Zahl auch jeweils den nächstgrößeren Bereich bezeichnet. So sind z. B. die Skalen aller analog anzeigenden Meßgeräte in der Weise konstruiert, daß eine Abbildung der unterscheidbaren Ablesebereiche auf einem Zahlenstrahl oder einem Zahlenkreis erfolgt.

Im Fall andersgearteter Unterschiede (Beispiel: Modellautosteuerung) kann man die Ordnungszahlen beliebig verteilen. Es ist aber auch hier aus merktechnischen Gründen sinnvoll, menschliche Denkgepflogenheiten nach Möglichkeit zu berücksichtigen und z. B. den Stillstand mit der Zahl 0 zu belegen.

Die Einbeziehung von ordnenden Zahlen könnte man als einfache Abzählung werten, wenn man die normale Reihenfolge der natürlichen Zahlen verwendet. Man kann diesen Vorgang aber auch

Tafel 1.5. Beispiele geordneter Zustandssysteme

Anzahl der Fahrzeuge je Minute	Zustandsbezeichnung	Bereichszahl
0	kein Verkehr	1
1 ... 10	geringer Verkehr	2
11 ... 30	mäßiger Verkehr	3
31 ... 100	starker Verkehr	4
101 ... 1000	sehr starker Verkehr	5

a) Beschreibung der Verkehrsdichte auf einer Kreuzung

Zustandsbezeichnung	Bereichszahl
Auto steht still	0
Auto fährt langsam	1
Auto fährt schnell	2
Auto fährt rechts herum	3
Auto fährt links herum	4
Auto fährt rückwärts	5

b) Zustandsbeschreibung für eine Spielzeugautosteuerung

Zustandsbezeichnung durch Angabe der Stromstärke	Bereichszahl
0 ... 10 mA	1
10 ... 20 mA	2
20 ... 30 mA	3
⋮	⋮
90 ... 100 mA	10

c) Definition von Stromstärkebereichen

so sehen, daß hier ein informationstragender Zustand durch einen anderen Zustand ersetzt wird. So wird im Beispiel a der Tafel 1.5 der Zustand, daß „je Minute 11 bis 30 Fahrzeuge die Kreuzung passieren", gleichgesetzt mit der Bezeichnung „mäßiger Verkehr" und gleichgesetzt mit der Zahl „3". Nach unserer Verabredung haben alle drei Angaben die gleiche Bedeutung, liefern demnach die gleiche Information.

Dabei sind aber unterschiedliche materielle Gebilde im Spiel. Am Anfang der Betrachtung steht der materielle Prozeß *Fahrzeugstrom*, der die erkennbare Eigenschaft, Anzahl der je Minute die Kreuzung passierenden Fahrzeuge, zu definieren gestattet. Aus der tatsächlichen Feststellung dieser Fahrzeuganzahl je Minute erfolgt die Zustandsbeschreibung durch die in Worten oder Schrift ausgedrückten Bezeichnungen „mäßiger Verkehr" oder „3". Die Information, die der tatsächliche Zustand liefert, ist also durch die Definition eines geordneten Zustandssystems übertragbar gemacht worden in andere Zustände. Eine solche Informationsübertragung auf andere Zustände ist ebenfalls eine *Codierung*, wobei das geordnete Zustandssystem gleichzeitig als *Codierungsvorschrift* anzusehen ist. Dabei können anstelle der Zahlen oder wörtlichen Bezeichnungen auch beliebige andere Zeichen oder physikalische Zustände verwendet werden.

Die Art des Informationsträgers wird stets nach Zweckmäßigkeitsüberlegungen ausgewählt. Meist will man die Information möglichst energiesparend, schnell und störungssicher verarbeiten; denn es gilt auch hier, Zeit und Kosten zu sparen, wenn man effektiv wirtschaften will.

Als besonders günstig hat sich die Informationsdarstellung mit Hilfe der elektrischen Energie erwiesen. Dafür sprechen folgende Gründe:

- Plötzliche Zustandsänderungen (Sprünge) von Stromstärke oder Spannung werden mit hoher Geschwindigkeit übertragen.
- Stromstärke und Spannung sind durch elektronische Schaltungen steuerbar, die wenig Material und Energie benötigen.
- Die elektrische Energie läßt sich in viele andere Energiearten umformen.

Bei der Wahl der zugelassenen Zustände von Stromstärke oder Spannung hat man die Anzahl von zwei Zuständen als beste Lösung ermittelt, da dann die beiden klar unterscheidbaren Fälle vorliegen

Strom fließt nicht ≙ 0
Strom fließt ≙ 1.

Wertet man die Zustände paralleler Stromleitungen aus, so kommt man zu 1/0-Verteilungen (Bitmustern), die den Dualzahlen gleichen. Da aber das Dualzahlensystem unendlich viele Zahlen aufweist, ist daraus erkennbar, daß auch mit zwei Zeichen (0 und 1) beliebig viele Zustände symbolisch darstellbar sind.

In diesem Zusammenhang ist es interessant, die Entwicklung der Informationsspeicherung im Laufe der Menschheitsgeschichte zu betrachten **(Tafel 1.6)** . Während man in der Vorzeit ungefähr ein Zeichen je Quadratzentimeter auf Steintafeln speichern konnte, werden heute mit Hilfe elektronischer oder magnetischer Speicher 10^5 bis 10^6 Zeichen je Quadratzentimeter festgehalten. Nach dem Prinzip der holografischen Speicherung (s. Abschn. 4.) erwartet man die Realisierung von 10^9 (eine Milliarde!) Zeichen je Quadratzentimeter.

Tafel 1.6. Grobe Abschätzung der Speicherdichten verschiedener Verfahren

Zeitangabe	Informationsspeicherung durch	Ungefähre Anzahl der Zeichen je Quadratzentimeter
Vorzeit	Steintafeln	1
	Tontafeln	3
	Holz, Elfenbein, Knochen	5
	Papyrusbogen, Pergament	5
Gegenwart	Papier (bedruckt)	20
	Kleinbildfilm	500
	Mikrofilm	2000
	Magnetplattenspeicher	$10^4 \ldots 10^6$
	Halbleiterspeicher	$10^5 \ldots 10^6$
	Magnetblasenspeicher	$10^5 \ldots 10^6$
	holografische Speicher	$10^7 \ldots 10^8$

(Bei den z. Z. angewendeten Verfahren ist die obere Grenze als noch nicht endgültig zu betrachten.)

1.3. Transformation der Informationsdarstellung von einem Zeichensystem in das andere

Informationen lassen sich durch beliebige Zeichensysteme darstellen.
Jedes Zeichensystem kann in jedes andere transformiert werden. Vorraussetzung dafür ist die Verabredung einer Zuordnungsvorschrift.

Neben den bereits auf S. 9 erwähnten Vorgängen Codierung und Decodierung kann man noch einen dritten Vorgang angeben: die *Umcodierung* oder *Codewandlung*. Dabei wird ein vorliegender Code in einen anderen überführt. Man spricht auch von der Transformation eines Zeichensystems in ein anderes.

Solche Umcodierungen werden z. B. notwendig, wenn automatische Informationsverarbeitungssysteme gekoppelt werden sollen, die nicht mit dem gleichen Zeichensystem arbeiten. Dabei gibt es zwei grundlegende Fälle:

a) Beide Systeme verfügen über die gleiche Anzahl von Zeichen.
b) Beide Systeme verfügen über eine ungleiche Anzahl von Zeichen.

Beim Fall a ist nur ein einfaches Zuordnungsproblem zu lösen. Haben wir z. B. bei einem System I die Zeichen

A, B, C, D

und bei System II die Zeichen

1, 2, 3, 4,

so sind die nach **Tafel 1.7c** folgenden Zuordnungen der Zeichen beider Systeme möglich, wenn stets alle Zeichen verwendet werden.

Tafel 1.7. Transformationsmöglichkeiten bei Systemen mit gleicher Zeichenzahl

A	1 2
B	2 1

a) bei zwei Zeichen

A	1 1 2 2 3 3
B	2 3 1 3 1 2
C	3 2 3 1 2 1

b) bei drei Zeichen

A	1 1 1 1 1 1	2 2 2 2 2 2	3 3 3 3 3 3	4 4 4 4 4 4
B	2 2 3 3 4 4	1 1 3 3 4 4	1 1 2 2 4 4	1 1 2 2 3 3
C	3 4 2 4 2 3	3 4 1 4 1 3	2 4 1 4 1 2	2 3 1 3 1 2
D	4 3 4 2 3 2	4 3 4 1 3 1	4 2 4 1 2 1	3 2 3 1 2 1

c) bei vier Zeichen

Unter Einbeziehung der Darstellungen nach Tafel 1.7a und b können Sie erkennen, daß die Anzahl der Transformationsmöglichkeiten sehr schnell zunimmt, wenn man die Anzahl n_z der Zeichen steigert. Die allgemeine Gesetzmäßigkeit für die Anzahl n der Umcodierungsmöglichkeiten lautet

$$n = n_z! \quad (n_z - \text{Fakultät})$$

$$n_z! = 1 \cdot 2 \cdot 3 \cdot 4 \ldots n_z.$$

Die Umcodierung gelingt auch dann, wenn die Anzahl der Zeichen des einen Systems kleiner als die des anderen Systems ist. Man ist aber dann im System mit weniger Zeichen gezwungen, die Zeichen mehrfach zu verwenden und die benötigten Unterschiede durch zeitliches oder räumliches Nach- bzw. Nebeneinander kenntlich zu machen.

Haben wir z. B. das System III mit den Zeichen

A, B, E, G, I, M, N, R, S, T

und das System IV mit den Zeichen 0, 1, so kann beispielsweise die Umcodierungstabelle nach **Tafel 1.8a** verwendet werden. Eine Wortbildung würde dann wie in Tafel 1.8b und eine Satzbildung wie in Tafel 1.8c aussehen.

Tafel 1.8. Transformationsbeispiele für zwei Systeme mit ungleicher Zeichenanzahl

A	B	E	G	I	M	N	R	S	T
1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	1	0	0	1
0	0	0	1	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	1

a) Umcodierungstabelle

B	R	I	N	G	E	N
0	0	1	1	0	1	1
1	0	0	1	0	1	1
0	0	1	1	1	0	1
0	1	0	0	0	0	0

b) Transformation eines Wortes

BRINGEN	SIE	BITTE	EINE	TASSE	TEE
0011011	111	01001	1111	01111	011
1001011	001	10111	1011	10001	111
0011101	010	01000	0110	00000	000
0100000	100	00110	0000	10110	100

c) Transformation eines Satzes

Die bekannte Transformation von Buchstaben nach dem Morsealphabet ist im **Bild 1.9** dargestellt. Da die Informationsübertragung mit Hilfe des Morsealphabets eine rein serielle Übertragung ist, daher recht lange dauert und auch ziemlich störanfällig ist, hat man für die Fernschreibtechnik und Datenübertragung andere Codearten entwickelt. Als Beispiel zeigt **Tafel 1.9** die Zuordnungen, die nach CCITT-Code Nr. 5 (ISO-Code) für Großbuchstaben verwendet werden.

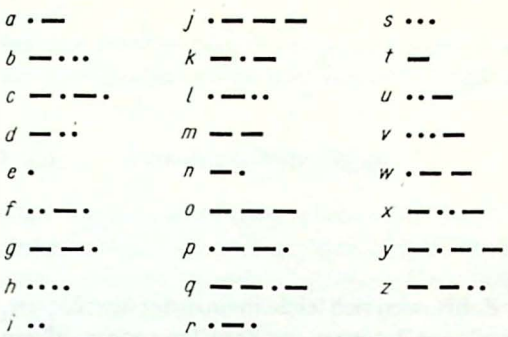
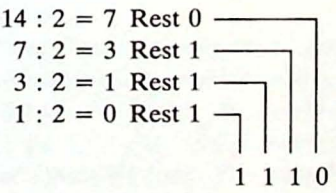


Bild 1.9
Transformation von Buchstaben nach dem Morsealphabet

Tafel 1.9. Codierung von Großbuchstaben nach dem CCITT-Code Nr. 5 (ISO-Code)

A	1	0	0	0	0	0	1	J	1	0	0	1	0	1	0	S	1	0	1	0	0	1	1
B	1	0	0	0	0	1	0	K	1	0	0	1	0	1	1	T	1	0	1	0	1	0	0
C	1	0	0	0	0	1	1	L	1	0	0	1	1	0	0	U	1	0	1	0	1	0	1
D	1	0	0	0	1	0	0	M	1	0	0	1	1	0	1	V	1	0	1	0	1	1	0
E	1	0	0	0	1	0	1	N	1	0	0	1	1	1	0	W	1	0	1	0	1	1	1
F	1	0	0	0	1	1	0	O	1	0	0	1	1	1	1	X	1	0	1	1	0	0	0
G	1	0	0	0	1	1	1	P	1	0	1	0	0	0	0	Y	1	0	1	1	0	0	1
H	1	0	0	1	0	0	0	Q	1	0	1	0	0	0	1	Z	1	0	1	1	0	1	0
I	1	0	0	1	0	0	1	R	1	0	1	0	0	1	0								

Zahlentransformation. Für die Transformation der Zahlen von einem Zahlensystem in das andere können Umwandlungsrechenvorschriften (Umwandlungsalgorithmen) angegeben werden. Dadurch erspart man sich umfangreiche Umcodierungstabellen.
So kann man bei der Transformation von Dezimalzahlen in Dualzahlen wie folgt vorgehen: Die gegebene Dezimalzahl wird durch zwei dividiert. Es entsteht ein ganzzahliger Quotient und ein Rest von null oder eins. Der Rest wird rechts unten notiert; der ganzzahlige Quotient wird erneut durch zwei dividiert. Der dabei entstehende Rest wird links neben dem ersten Rest notiert. Das wird in dieser Form fortgesetzt, bis als ganzzahliger Quotient die Null entsteht.
Beispiel: Umwandlung der Dezimalzahl 14 in eine Dualzahl



Vielleicht versuchen Sie jetzt selbst, die Dezimalzahl 125 in eine Dualzahl umzuwandeln! Das Ergebnis ist Ihnen bereits aus Bild 1.7 bekannt.

Bei der Umwandlung von Dezimalzahlen in Oktal- bzw. Hexadezimalzahlen wird analog vorgegangen, aber jeweils durch 8 bzw. 16 dividiert. Sehen Sie sich dazu die beiden folgenden Beispiele an:

$$\begin{array}{rl} 125 : 8 = 15 & \text{Rest } 5 \\ 15 : 8 = 1 & \text{Rest } 7 \\ 1 : 8 = 0 & \text{Rest } 1 \\ & \quad 1\ 7\ 5 \end{array}$$

Ergebnis: 125 = 175 Q

$$\begin{array}{rl} 125 : 16 = 7 & \text{Rest } 13 \\ 7 : 16 = 0 & \text{Rest } 7 \\ & \quad 7\ D \end{array}$$

Ergebnis: 125 = 7D H.

Duale, oktale und hexadezimale Darstellungen einer Zahl lassen sich leicht ineinander überführen. Die Ursache hierfür liegt in ihren Basiszahlen, die jeweils aus Potenzen der Zahl 2 mit ganzzahligen Exponenten gebildet sind.
Der Übergang von der dualen zur oktalen bzw. hexadezimalen Darstellung wird durch Zusammenfassung von jeweils drei bzw. vier Bitstellen, von rechts aus beginnend, durchgeführt. So wird

$$125 = \underbrace{1}\ \vdots\ \underbrace{111}\ \vdots\ \underbrace{101}\ \text{B} = \underbrace{111}\ \vdots\ \underbrace{1101}\ \text{B}$$

zu 1 7 5 Q bzw. 7 D H.

Natürlich kann diese Umwandlung auch in der umgekehrten Richtung vollzogen werden.
Die Transformation von allen anderen Zahlensystemen in das Dezimalsystem kann über die Summierung der einzelnen Stellenanteile erfolgen, die jeweils aus Koeffizient multipliziert mit der Stellenwertpotenz bestehen. So wird z. B.

$$\begin{aligned} 110011\ \text{B} &= 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 32 + 16 + 2 + 1 \\ &= 51. \end{aligned}$$

Man kann aber auch die weiter oben gezeigten Rechnungen rückwärts ausführen. Wir wollen diesen Rechengang für die Transformation der Dualzahl 110011 in das Dezimalsystem demonstrieren. Man beginnt mit dem Koeffizienten des höchstewertigen Bit, multipliziert diesen mit zwei und zählt den Koeffizienten der nächstniederen Bitstelle hinzu. Dieses Teilergebnis wird wieder mit zwei multipliziert und der Koeffizient der nächstfolgenden Bitstelle addiert. Das geht so weiter, bis alle Bitstellen erfaßt sind.

$$\begin{array}{l} 1\ 1\ 0\ 0\ 1\ 1 \quad 1 \cdot 2 + 1 = 3 \\ \qquad\qquad\quad 3 \cdot 2 + 0 = 6 \\ \qquad\qquad\quad 6 \cdot 2 + 0 = 12 \\ \qquad\qquad\quad 12 \cdot 2 + 1 = 25 \\ \qquad\qquad\quad 25 \cdot 2 + 1 = 51. \end{array}$$

1.4. Informationsverarbeitung

Unter Informationsverarbeitung versteht man alle Prozesse, bei denen Informationen gesammelt, gespeichert, ausgewählt, verdichtet und mit Hilfe spezieller Verfahren ausgewertet werden. Diese Prozesse müssen sehr exakt geplant und in allen Einzelheiten durchdacht werden.

Für den Entwurf eines solchen Prozesses bedient man sich verschiedener Methoden. Eine davon soll im folgenden vorgestellt werden: das Arbeiten mit dem *Programmablaufplan* (PAP).

1.4.1. Programmablaufplan

Ein Programmablaufplan ist ein grafisches Schema, das die einzelnen Prozeßschritte in ihrer Aufeinanderfolge und in ihrer gegenseitigen Wechselwirkung zeigt und sich vor allem für die Planung automatisch ablaufender Prozesse eignet. **Tafel 1.10** zeigt eine Auswahl der nach TGL 22451 festgelegten Sinnbilder.
Im **Bild 1.10** ist ein Beispiel eines Programmablaufplans gegeben. Versuchen Sie zu ergründen, was bei der Abarbeitung dieses Programmablaufplans vor sich geht, und schreiben Sie Ihr Ergebnis in Satzform nieder!
Fassen Sie dabei x als Variable auf. *Lesen Sie jetzt noch nicht den Text neben dem Bild 1.10, wenn Sie sich überprüfen wollen.*

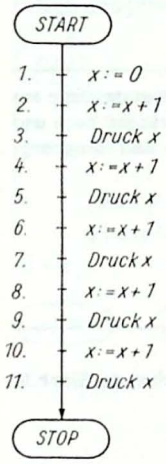


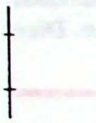
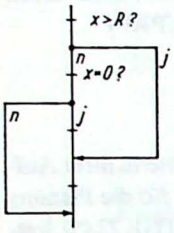
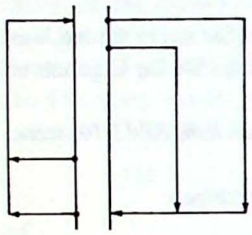
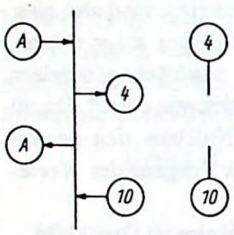
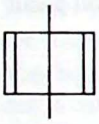
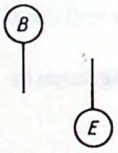
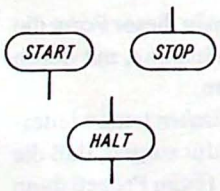
Bild 1.10. Beispiel eines einfachen Programmablaufplans

Der Programmablaufplan läßt sich verbal wie folgt beschreiben:

1. Die Variable x wird zu Null gemacht.
2. Die Variable x ergibt sich aus dem bisherigen Wert plus eins, wird also eins. Beachten Sie dabei bitte folgendes: Die Formulierungen $x := 0$ bzw. $x := x + 1$ sollen keine Gleichheit im üblichen Sinn ausdrücken, sondern werden gelesen „x ergibt sich aus Null" bzw. „x ergibt sich aus $x + 1$". Damit soll beschrieben werden, daß die Variable x nun durch Null bzw. den um eins erhöhten Wert ersetzt wird. Es wird also damit auf den Vorgang der Wertebildung hingewiesen.
3. Der Wert von x wird gedruckt. Damit erscheint die Zahl eins im Druckbild.
4. $x := x + 1$. Die Variable x wird wiederum um eins erhöht und nimmt damit den Wert zwei an.
5. Druck von x. Die Zahl zwei wird gedruckt.
6. Die Variable x wird wiederum um eins erhöht und wird zu drei.
7. Druck von x. Es wird die Zahl drei gedruckt.

Das geht so weiter, bis unter 11. die Zahl fünf im Druckbild erscheint. Dann geht das Programm in den STOP-Zustand.
Zusammenfassend kann man die Wirkungsweise des Programms wie folgt beschreiben:
„Es werden die Zahlen von eins bis fünf gebildet und ausgedruckt."
Sie können sich nun vorstellen, wie lang ein solches Programm würde, wenn man in dieser Form die Zahlen von 1 bis 100 bilden und drucken sollte. Es gibt aber eine einfache Maßnahme, mit deren Hilfe man die Notierung eines derartigen Programms wesentlich verkürzen kann.
Man muß dazu anstelle des bisher gestreckten Programmablaufplans einen *zyklischen* (sich wiederholenden) Programmablaufplan verwenden. Mit anderen Worten: Man muß dafür sorgen, daß die sich wiederholenden Elemente nur einmal geschrieben, aber wiederholt gelesen (beim Prozeß dann

Tafel 1.10. Auszugsweise Darstellung von Sinnbildern für Programmablaufpläne nach TGL 22451

Darstellung	Benennung und Erklärung
	Zuordnung von Operationen (Anweisungen) zur Programmlinie, einzeln oder zusammengefaßte Befehle in mathematischer oder textlicher Form, grundsätzlich rechts neben die vertikale Programmlinie schreiben
	Verzweigen der Programmlinie nach rechts oder links, Zusammenführen der Programmlinien von rechts oder links Sprung im Ergebnis einer Vergleichsoperation
	Zusammenführen mehrerer Programmlinien von links und rechts. Die vertikalen Parallellinien müssen beim Verzweigen und Zusammenführen von rechts für eventuelle Beschriftungen ausreichend Abstand voneinander haben, andernfalls sind Konnektoren zu verwenden.
	Zeichenkonnektor für Unterbrechung und Fortsetzung der Programmlinie aus zeichnerischen Gründen. Mit Pfeilspitze bei Ab- und Einsprüngen nach und von links und rechts. Zusammengehörige Konnektoren (Ab- und Einsprungstellen) müssen die gleiche Bezeichnung erhalten
	Unterprogramm, Programmteil Der Name des Unterprogramms, Programmteils, ist rechts neben das Sinnbild zu schreiben
	Beginn und Ende eines Unterprogramms, Programmteils, bei detaillierter Darstellung außerhalb der Programmhauptlinie
	Programmanfang, Programmunterbrechung, Programmende

wiederholt abgearbeitet) werden. Ein zyklischer Programmablaufplan, der die Bildung und den Druck der Zahlen von 1 bis 100 beschreibt, ist im **Bild 1.11** dargestellt.

Versuchen Sie nun, sich diesen zyklischen Programmablaufplan zu erklären! Sie erkennen von dem gestreckten Programmablaufplan die Operationen $x := 0$, $x := x + 1$ und Druck x wieder. Zusätzlich erscheint die Frage $x = 100$? Solange diese mit nein (n) beantwortet wird, wiederholt sich der Programmablaufplan in der Form $x := x + 1$ und Druck x. Erst wenn $x = 100$ ist, wird die Frage bejaht, und das Programm läuft in den STOP-Zustand.

Das zyklische Programm wird durch eine *Verzweigungsstelle* erreicht. Die Programmierung einer solchen Verzweigungsstelle führt später im eigentlichen Programm zu einem *bedingten Sprung*. Sprung deshalb, weil zu einer bereits abgearbeiteten Anweisung zurückgegangen (zurückgesprungen) wird, und bedingter Sprung, weil bei Abarbeitung des Programms durch Prüfung der Bedingung $x = 100$? entschieden wird, ob der Sprung zur Ausführung gelangt oder nicht.

Man erkennt, daß der Automat eine sehr exakte und für uns ungewohnt umständliche Arbeitsanweisung benötigt. Für den Menschen kann man den Auftrag kürzer fassen und ansagen: „Schreiben Sie bitte die Zahlen von 1 bis 100!" Auch einen Automaten kann man mit einem derartigen Satz programmieren, wenn man den entsprechenden technischen Aufwand treibt. Das bedeutet, daß der Automat das betreffende Programm schon eingespeichert hat und der Satz „Schreiben Sie bitte die Zahlen von 1 bis 100" nur zur Erkennung des Auftrags eingegeben werden muß, um damit das zugehörige Programm aussuchen und starten zu können. Auf jeden Fall muß der Programmierer die Zergliederung eines automatischen Prozesses bis in die dem Automaten verständlichen Befehle beherrschen. Bis zu welchem Grad der Vereinzelung diese Befehle aufzugliedern sind, ist eine Frage des jeweiligen Entwicklungsstands.

Bild 1.11. Programmablaufplan für Bildung und Druck der Zahlen von 1 bis 100

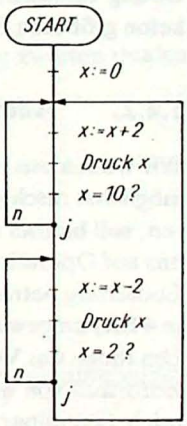


Bild 1.12. Beispiel eines Programmablaufplans mit mehrfacher Verzweigung

*Versuchen Sie nun zur Übung, die Wirkung des im **Bild 1.12** dargestellten Programmablaufplans in zusammenfassender Form anzugeben!*

Ergebnis: Es werden alle geraden Zahlen von zwei bis zehn aufwärts zählend gebildet und gedruckt. Danach werden alle geraden Zahlen von acht bis zwei abwärts zählend gebildet und ausgedruckt. Danach wiederholt sich der Vorgang periodisch. Damit erhält man nachtstehende Zahlenfolge:

2, 4, 6, 8, 10, 8, 6, 4, 2, 4, 6, 8, 10, 8, 6, 4, 2 usw.

Mit diesen Beispielen sind Ihnen einfache überschaubare Programmablaufpläne gezeigt worden. Es können aber nicht nur Anweisungen zur Durchführung von Rechenoperationen, sondern auch andere Automatenoperationen mit Hilfe eines Programmablaufplans dargestellt werden. Deshalb wollen wir zum Abschluß dieser Betrachtungen noch ein einfaches Beispiel für eine *Maschinensteuerung* an einer automatischen Fertigungslinie kennenlernen **(Bild 1.13)**.

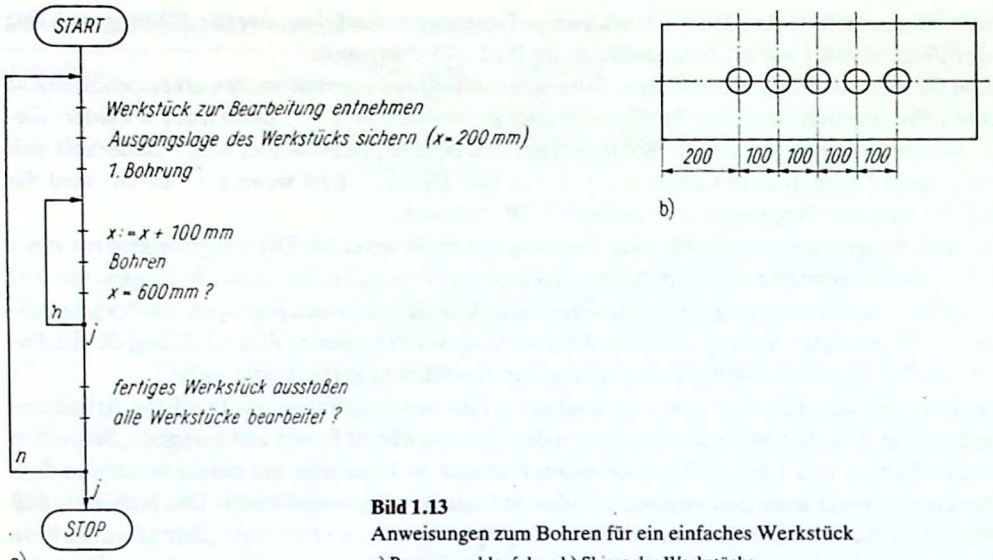


Bild 1.13
Anweisungen zum Bohren für ein einfaches Werkstück
a) Programmablaufplan; b) Skizze des Werkstücks

In ein Werkstück sollen im Abstand von 100 mm zeitlich nacheinander fünf Löcher gebohrt werden. Mit der Variablen x soll jeweils die Koordinate der einzelnen Bohrlochzentren bezeichnet werden, wobei die Mitte des ersten Loches bei $x = 200$ mm liegen soll. Da wir jetzt schon einige Übung besitzen, dürfte das selbständige Durchdenken der Darstellung im Bild 1.13 für Sie nun keine größeren Schwierigkeiten mehr bereiten.

1.4.2. Addition und Subtraktion

Wir wollen uns nun der Lösung arithmetischer Probleme zuwenden. Nach dem Grundsatz, Ihnen möglichst rasch (ohne nach Vollständigkeit zu streben) die wichtigsten Zusammenhänge zu erklären, soll hierbei nicht auf Multiplikation und Division eingegangen werden. Außerdem wollen wir uns auf Operationen an reinen Dualzahlen beschränken. Wir wollen also ebenfalls nicht die BCD-Codierung betrachten, bei der die Ziffern der einzelnen Stellen einer Dezimalzahl in Tetraden (zu je 4 Bit) umgewandelt werden.
Um Ihnen das Verständnis noch weiter zu erleichtern, wollen wir für diesen Abschnitt ein Dualzahlenformat von nur vier Bitstellen verwenden. Damit ist nur ein stark eingeschränkter Zahlenbereich darstellbar. Die dabei gewonnenen Einsichten lassen sich aber ohne größere Schwierigkeiten auch auf größere Zahlwortbreiten, wie 8 oder 16 Bit, übertragen.

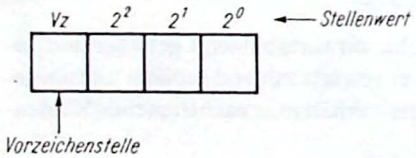


Bild 1.14
Stellenwertschema für eine vierstellige Dualzahl mit Vorzeichenstelle

Zahlwortdefinition. Das von uns verwendete Zahlwort soll, wie im **Bild 1.14** dargestellt, definiert werden. Neu ist dabei, daß die höchste Bitstelle nicht als Stellenwert, sondern als Vorzeichen definiert wird. Dazu verabreden wir:

- Vorzeichenbit = 0 entspricht einer positiven Zahl
- Vorzeichenbit = 1 entspricht einer negativen Zahl.

Betrachten wir zunächst einmal nur die mit diesen Festlegungen darstellbaren positiven Zahlen, so erhalten wir die in **Tafel 1.11** gezeigte Übersicht.

Tafel 1.11
Positive Dualzahlen von null bis sieben

Dezimalzahl	Dualzahl
0	0000
+ 1	0001
+ 2	0010
+ 3	0011
+ 4	0100
+ 5	0101
+ 6	0110
+ 7	0111

Addition. An einigen Beispielen soll die Addition mit Dualzahlen aus Tafel 1.11 erläutert werden. Zunächst wollen wir die duale Eins mit der dualen Zwei addieren.

0001	1
+ 0010	+ 2
0011	3

Es gilt: 1 + 0 = 1.

Das ging problemlos. Vollführen wir nun die Addition der dualen Eins mit einer zweiten dualen Eins, so erhalten wir einen Übertrag in die zweite Stelle.

Übertrag 1	
0001	1
+ 0001	+ 1
0010	2

Es gilt: 1 + 1 = 0 + Übertrag 1.

Am Beispiel der Addition der dualen Drei mit sich selbst soll gezeigt werden, wie drei untereinanderstehende Ziffern vom Wert eins (1 + 1 + Übertrag 1) addiert werden.

Übertrag 11	
0011	3
+ 0011	+ 3
0110	6

Es gilt: 1 + 1 + Übertrag 1 = 1 + Übertrag 1.

Wir wollen diese Ergebnisse verallgemeinern. Dazu bezeichnen wir den Koeffizienten einer betrachteten Bitstelle der ersten Zahl mit a_n und den Koeffizienten der gleichen Stelle bei der zweiten Zahl mit b_n und einen möglichen Übertrag von der vorhergehenden Stelle mit $\ddot{u}_{n-1}$. Kombiniert man alle möglichen Wertebelegungen für a_n, b_n und $\ddot{u}_{n-1}$, so erhält man für die Stellensumme s_n und den Übertrag zur nächsten Stelle $\ddot{u}_n$ die in der **Tafel 1.12** angegebene Zuordnung.

Mit Hilfe dieser Tafel wollen wir noch folgende Aufgaben rechnen:

```
      0010    2
    + 0100  + 4
      0110    6

      0010    2
    + 0101  + 5
      0111    7

Übertrag 011
      0010    2
    + 0110  + 6
  !→1000    8

Übertrag 011
      0010    2
    + 0111  + 7
  !→1001    9
```

Tafel 1.12. Zuordnungstabelle für die Addition zweier Binärstellen bei Berücksichtigung eines Übertrags

$\ddot{u}_{n-1}$	b_n	a_n	s_n	$\ddot{u}_n$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Im Prinzip sind die beiden letzten Berechnungen richtig durchgeführt worden, ergeben aber doch ein nicht erlaubtes Ergebnis, da wir die vierte Stelle für das Vorzeichen vorgesehen haben. Im *Vorzeichenbit* müßte aber eine Null stehen, da das Ergebnis bei der Addition zweier positiver Zahlen wieder eine positive Zahl sein muß. Die Belegung der Vorzeichenbitstelle durch eine Eins aus der Rechenoperation (Übertrag) heraus entstpricht einem *Überlauf*. Dieser Überlauf muß unbedingt signalisiert werden, damit das falsche Ergebnis nicht weiterverarbeitet wird.

Subtraktion. Die Subtraktion könnten wir nun in ähnlicher Art und Weise wie die Addition einführen. Dabei würden wir eine neue Rechenvorschrift erhalten, die ähnlich wie die in Tafel 1.12 für die Addition aussähe. Für die technische Realisierung müßte dann anstelle einer *Addierstufe* (Adder) noch eine *Subtrahierstufe* aufgebaut werden.

Diese Subtrahierstufe kann man jedoch einsparen, wenn man die Subtraktion in eine Addition umwandelt.

Aus $y = a - b$ (Differenz y = Minuend a − Subtrahend b) macht man $y = a + (-b)$.

Man muß also den Subtrahenden in eine negative Zahl verwandeln. Dazu muß die Zahl b durch ihr Zweierkomplement ersetzt werden.

- Das Zweierkomplement einer Dualzahl wird gebildet, indem man entweder

  a) alle Ziffern *invertiert* (umkehrt), d. h. 0 in 1 und 1 in 0 umschreibt, und anschließend in der niedrigsten Stelle eine Eins addiert oder

  b) alle Ziffern nach der ersten Eins, die von rechts aus gelesen erscheint, invertiert.

Beispiel 1. *Gegeben:* 3 = 0011 B

```
Anwendung von Verfahren a
Invertieren jeder Stelle      1100
Addition einer Eins         +    1
                              ----
Zweierkomplement der gegebenen Zahl  1101 ≙ −3
```

Beachten Sie bitte, daß die duale negative Drei im Zweierkomplement nicht einfach dadurch entsteht, daß man die duale positive Drei nimmt und im höchstwertigen Bit (in der Vorzeichenstelle) eine Eins setzt!

Beispiel 2. *Gegeben:* 6 = 0110 B

Verfahren a	1. Schritt	1001
	2. Schritt	+ 1
	Zweierkomplement	1010 ≙ −6

Verfahren b

Die rechte Null und die erste Eins von rechts bleiben erhalten. Die restlichen beiden Ziffern werden invertiert

	invertieren	erhalten
Gegebene Zahl	0 1	1 0
Zweierkomplement	1 0	1 0

Stellt mann alle möglichen positiven und negativen Zahlen für die von uns vereinbarte Zahlwortbreite von 4 Bit zusammen, erhält man Tafel 1.13. Eine Ausnahmeerscheinung in dieser Tafel stellt die Zahl 1000 B dar, die als −8 eingestuft worden ist. Wir werden im folgenden einsehen, daß diese Einstufung durchaus logisch und konsequent ist.
Sehen wir uns nun an, welche Ergebnisformen möglich sind, wenn wir mit dem gesamten nach **Tafel 1.13** definierten Zahlenbereich arbeiten.
Addiert man je eine positive und eine negative Zahl unserer Tafel, entstehen durchweg Ergebnisse, die wiederum im Bereich der Tafelzahlen liegen müssen, also keine Unregelmäßigkeiten aufweisen dürften. Hierzu wollen wir uns die folgenden drei Beispiele ansehen:

Übertrag	0000	
	1011	− 5
	+ 0100	+ 4
	1111	− 1
Übertrag	1111	
	1011	− 5
	+ 0101	+ 5
	0000	0
Übertrag	1110	
	1011	− 5
	+ 0110	+ 6
	0001	+ 1

Tafel 1.13. Mit vier Bitstellen darstellbare ganze Dualzahlen von −8 bis +7

Dezimalzahl	Dualzahl
+ 7	0111
+ 6	0110
+ 5	0101
+ 4	0100
+ 3	0011
+ 2	0010
+ 1	0001
0	0000
− 1	1111
− 2	1110
− 3	1101
− 4	1100
− 5	1011
− 6	1010
− 7	1001
− 8	1000

Aus den beiden letzten Beispielen ersieht man, daß doch eine Besonderheit auftritt. Es gibt einen Übertrag über die definierte Zahlwortbreite hinaus. Wertet man jedoch nur die vereinbarten vier Stellen, so erhält man das richtige Ergebnis.
Von vornherein anders sieht es aus, wenn man zwei negative Zahlen der Tafel 1.13 addiert oder von einer negativen Zahl eine positive Zahl subtrahiert. Hierbei ist ohne weiteres erkennbar, daß au-

ßerhalb des vereinbarten Zahlenbereichs liegende Ergebnisse möglich sind. Sehen wir uns dazu folgende Beispiele an:

Übertrag	1111	
	1011	− 5
	+ 1111	+ (− 1)
	1010	− 6

Bei der Addition von 1011 ≙ −5 und 1111 ≙ −1 erhält man 1010 und einen Übertrag in die fünfte Stelle. Dieses Ergebnis entspricht der −6, wenn man die zusätzliche Übertragsstelle (fünftes Bit) nicht wertet.

Übertrag	1111	
	1011	− 5
	+ 1101	+ (− 3)
	1000	− 8

Bei der Addition von 1011 ≙ −5 und 1101 ≙ −3 erhält man als Ergebnis der ersten vier Bit 1000 ≙ −8 und einen Übertrag in die fünfte Stelle. Die Definition der 1000 als −8 erweist sich folglich als richtig.

Übertrag	1000	
	1011	− 5
	+ 1100	+ (− 4)
	0111	− 9

Bei der Addition von 1011 ≙ −5 und 1100 ≙ −4 erhält man bei Wertung der ersten vier Bit als Ergebnis 0111, was +7 entspricht, also falsch ist.
Auch beim nächsten Beispiel 1001 ≙ −7 plus 1001 erhält man bei Wertung der ersten vier Bit ein falsches Ergebnis.

Übertrag	1001	
	1001	− 7
	+ 1001	+ (− 7)
	0010	−14

Wie kann nun ein Automat die falschen Ergebnisse von den richtigen unterscheiden? *Vielleicht überprüfen Sie selbst, ehe Sie weiterlesen, ob Sie eine Erkenntnismethode finden.*
Lösung: Man muß die Überträge in das Vorzeichenbit und in die zusätzliche fünfte Bitstelle beachten. Bei den als richtig wertbaren Ergebnissen sind diese Überträge beide gleich und bei den falschen Ergebnissen ungleich.
Diese Feststellung kann man auch auf die Bewertung der Ergebnisse bei der Addition positiver Zahlen übertragen. Dort entstehen richtige Ergebnisse, wenn die Überträge in das vierte und fünfte Bit jeweils gleich Null sind, und falsche Ergebnisse, wenn die Überträge in diese Bitstellen ungleich sind. *Sehen Sie sich dazu noch einmal diese Aufgaben an!*
Um diese Aussagen noch weiter zu verallgemeinern, d. h. für Dualzahlen mit beliebig vielen definierten Bitstellen anzuwenden, nennen wir den bisherigen Übertrag zum fünften Bit den Ausgabeübertrag $\ddot{u}_a$ und den bisherigen Übertrag in das vierte Bit den Übertrag in das Vorzeichenbit $\ddot{u}_v$.

Dann lautet die allgemeine Aussage:

- Bei der Addition und Subtraktion von vorzeichenbehafteten Dualzahlen kann man ein entstandenes ungültiges Ergebnis daran erkennen, daß die Überträge $\ddot{u}_a$ und $\ddot{u}_v$ ungleich sind. Sind beide Überträge gleich (0, 0 oder 1, 1), liefern die definierten Bitstellen das richtige Ergebnis.

Bei Vorliegen von solcherart entstandenen ungültigen Ergebnissen spricht man auch von Überlauf (engl.: overflow).

1.4.3. Automatische Zustandserkennung mit Folgereaktion

Wir wissen: Zustandssysteme werden aus erkennbaren Eigenschaften materieller Gebilde oder materieller Prozesse heraus definiert, indem man abgrenzbare Merkmalsunterschiede angibt.

Zur automatischen Zustandserkennung benötigt man Meß- und Erkennungseinrichtungen, die die definierten *Merkmalsunterschiede* feststellen können. Zur prozeßoptimalen Folgereaktion auf den jeweils ermittelten Zustand werden Rechenautomaten (vorzugsweise Mikrorechner) verwendet, die mit Hilfe ihrer Rechenprogramme die geeigneten Schlußfolgerungen ziehen und die Betätigung der entsprechenden Stelleinrichtungen veranlassen können.
Betrachten wir hierzu einige Beispiele.

Beispiel 1. In einem Betrieb zur Herstellung von Transistoren sollen die fertigen Bauelemente nach Stromverstärkungswerten in drei Gruppen sortiert werden.

Gruppe	Bereich der Stromverstärkung
A	30 ... 100
B	größer 100 ... 300
C	größer 300

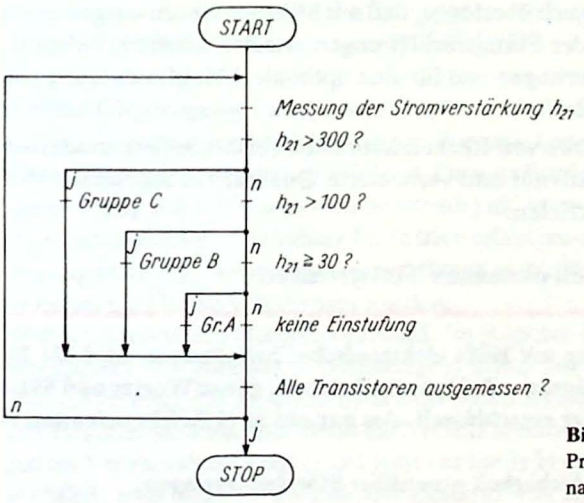


Bild 1.15
Programmablaufplan für die Sortierung von Transistoren nach Stromverstärkungswerten

Dazu muß die Stromverstärkung jedes Transistors gemessen werden, wozu spezielle Meßverfahren existieren. Danach muß ein jeder Transistor in eine der drei Gruppen eingeordnet werden. Das kann mit einem Rechenautomaten realisiert werden, dessen Rechenprogramm nach dem im **Bild 1.15** dargestellten Programmablaufplan gestaltet ist.

Nach erfolgter Einordnung ist der Vorgang der automatischen Zustandserkennung beendet. Diese *Zustandserkennung* oder auch Gruppenzuteilung kann zur Realisierung weiterer technischer Maßnahmen (Folgereaktionen) ausgewertet werden. So kann beispielsweise der Kennbuchstabe für die Stromverstärkung auf den Transistor aufgedruckt werden und beim Weitertransport des gemessenen Transistors dafür gesorgt werden, daß er in das zugehörige Sortierfach eingeordnet wird. Außerdem können die Meßergebnisse aller Transistoren einer Fertigungscharge statistisch ausgewertet werden, so daß die Technologen Hinweise für die Qualität des Herstellungsprozesses erhalten. Bei einem Prozeß mit hohem Automatisierungsgrad kann ein Rechenautomat mit entsprechend kompliziertem Programm sogar selbständig die Gesamtheit aller Meßergebnisse analysieren und in folgerichtige Entscheidungen zur *Optimierung* der Verfahrensschritte umsetzen.

Beispiel 2. Wir wollen einen Rechenautomaten betrachten, der einen automatischen Prozeß steuert und überwacht. Die Prozeßsteuerung dient immer dem Zweck, die vorgegebenen Produktionsziele bestmöglich zu erreichen. Die Überwachung wird so organisiert, daß zum einen die Einhaltung der Produktkennwerte, zum anderen aber die Fehlerfreiheit der Funktionsgruppen des produzierenden Systems unter Einsatz entsprechender Meßgeräte überprüft werden.
Wird festgestellt, daß irgendein Teilsystem nicht mehr einwandfrei arbeitet, so kann der Automat mit Hilfe von *Diaganoseprogrammen* den Fehler erkennen und die zur Fehlerbeseitigung notwendigen Maßnahmen einleiten oder signalisieren. Der Rechenautomat kann außerdem dazu beitragen, daß mit Energie und Rohstoffen sparsam gewirtschaftet wird.

Beispiel 3. Mit Hilfe einer mikroprozessorgesteuerten Zusatzeinrichtung kann ein Telefonapparat wichtige Rufnummern speichern und einen Anruf bis zur Herstellung der gewünschten Verbindung selbständig durchführen. Das Speichern der wichtigen Rufnummern bringt für den Anrufer den Vorteil, daß er anstelle sechs- bis zwölfstelliger Nummern nur noch zwei Ziffern vorzugeben braucht, wenn er die Anzahl dieser wichtigen Rufnummern auf weniger als 100 beschränkt. Dazu muß er sich eine *Vereinbarungsliste* anfertigen, auf der diese Rufzahlen und die jeweils zugehörige *Ordnungszahl* notiert werden. Nach Einspeicherung der Liste in den Automaten braucht er dann für den Anruf eines dieser Teilnehmer nur noch dessen zweistellige Listennummer vorzugeben und den Startknopf zu drücken. Die weitere Arbeit übernimmt dann die Automatik, die auch bei Erkennen des Besetztzeichens in gewissen Zeitintervallen die Anrufversuche fortsetzen kann.
In Ergänzung dieses Beispiels kann man sich auch überlegen, daß mit Mikrorechnern ausgestattete Vermittlungsämter eine höhere Auslastung der Fernsprechleitungen erzielen können, indem sie für eine Koordinierung der Gesprächsanforderungen und für eine optimale Ablaufssteuerung sorgen. Zusammenfassend kann festgestellt werden:

- Die Automatisierung von Prozessen mit Hilfe von Rechenautomaten (insbesondere modernen Mikrorechnern) ermöglicht, größere Effektivität und verbesserte Qualität bei sparsamer Verwendung von Energie und Rohstoffen zu erzielen.

1.4.4. Informationsaustausch zwischen einzelnen Teilsystemen

Wir wissen: Bei der Informationsverarbeitung mit Hilfe elektronischer Schaltungen wird die Binärzeichendarstellung benutzt. Alle Informationen – Zahlen, Buchstaben, ganze Wörter und Sätze – erscheinen dann symbolisch in einem Muster verschlüsselt, das nur aus zwei Zeichenelemten (0 und 1) besteht.
Wir kennen auch den Grund dafür: optimale Sicherheit gegenüber Störeinwirkungen.

Leider hat diese Festlegung auch nachteilige Konsequenzen. Vergleichen wir nur die Anzahl der Stellen bei der Darstellung einer Zahl im Dezimal- und Binärsystem. So benötigt die Zahl 6375 vier Stellen im Dezimalsystem und dreizehn Stellen im Binärsystem.

1100011100111 B = 6375

Ähnlich sieht es bei der Buchstabendarstellung aus. Wir haben gesehen, daß ein Buchstabe nach dem CCITT-Code Nr. 5 sieben Binärstellen erfordert, wobei man allerdings berücksichtigen muß, daß mit diesen sieben Bitstellen auch noch Kleinbuchstaben, Ziffern und Sonderzeichen erfaßt werden. Damit man gewisse Übertragungsfehler erkennen kann, fügt man vielfach ein weiteres Bit mit der Absicht hinzu, die Anzahl der mit 1 besetzten Bit geradzahlig oder ungeradzahlig zu machen. Dieses Zusatzbit nennt man *Paritätsbit*. Durch Kontrolle der vorgegebenen Parität kann man z. B. bei Lochstreifeneingaben einfache Lochungs- oder Lesefehler erkennen.
Die große Stellenanzahl, die durch die Binärkodierung entsteht, zwingt den Entwickler von Informationsverarbeitungssystemen dazu, alle Systementwürfe genau zu durchdenken und nach optimalen Lösungen zu suchen. Macht er das nicht, können solche Anlagen vergleichsweise zu teuer werden und zu geringe Verarbeitungsgeschwindigkeiten aufweisen.
Wir wollen hier nicht tiefer in diese Problematik eindringen. Wir nehmen sie aber zum Anlaß, um uns einmal grundlegende Gedanken darüber zu machen, wie *binär codierte* Informationen zwischen einzelnen Systemen ausgetauscht werden können.
Dazu muß eine Übertragung vorgenommen werden, die auf einer Leitung oder auf mehreren Leitungen gleichzeitig erfolgen kann. Wählen wir als Beispiel eine achtstellige Dualzahl, und definieren wir die Wertigkeiten und die Bezeichnungen der Speicherplätze der einzelnen Stellen wie im **Bild 1.16**. Für eine solche zusammengehörige Anordnung von acht Bitstellen verwendet man auch den Begriff Byte (soviel wie Wort) und hat die Vereinbarung getroffen

1 Byte = 8 Bit.

Das höchstwertige Bit wird auch mit MSB (most significant bit) und das niedrigstwertige Bit auch als LSB (least significant bit) bezeichnet.

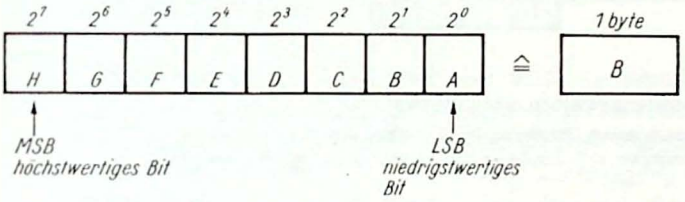


Bild 1.16
Speicherplatzschema für eine achtstellige Dualzahl

Serielle Übertragung. Hierbei wird das Byte Bit für Bit von der sendenden Baugruppe zur empfangenden Baugruppe geschickt. Dieser Vorgang kann im Rhythmus eines Taktes (synchron) oder taktunabhängig (asynchron) erfolgen. Da zum Senden und Empfangen eine wenigstens zeitweilige Speicherung der Informationen notwendig ist, verwendet man Speicherbausteine, die als *Register* bezeichnet werden. Die **Bilder 1.17a bis e** erläutern den Vorgang einer seriellen Informationsübertragung über sog. *Schieberegister*, bei denen je Auslöseimpuls (Schiebeimpuls) die einzelnen Bit jeweils um eine Stelle verschoben werden.
Bild 1.17a zeigt den Ausgangszustand. Im Register 1 (Sender) ist die gegebene Dualzahl 10011101 gespeichert. Im Register 2 (Empfänger) sollen alle Speicherplätze den Zustand Null aufweisen. Bild 1.17b zeigt die Situation nach dem ersten Auslöseimpuls. Alle Speicherstelleninhalte der beiden Register sind um eine Stelle nach rechts verschoben worden. Der vorher an der Stelle A des Registers 1 vorhandene Zustand ist jetzt zur Stelle H des Registers 2 übertragen worden. In die Stelle H (MSB) vom Register 1 wurde der Zustand Null eingeschoben.
Bild 1.17c zeigt den Zustand der beiden Register nach dem zweiten, Bild 1.17d den Zustand nach dem dritten Auslöseimpuls. Der Endzustand wird nach acht Auslöseimpulsen erreicht. Dann enthält das Register 1 nur noch Nullen und Register 2 den vorher in Register 1 gespeicherten Zustand. Damit ist die gegebene Dualzahl vom Register 2 übernommen worden.

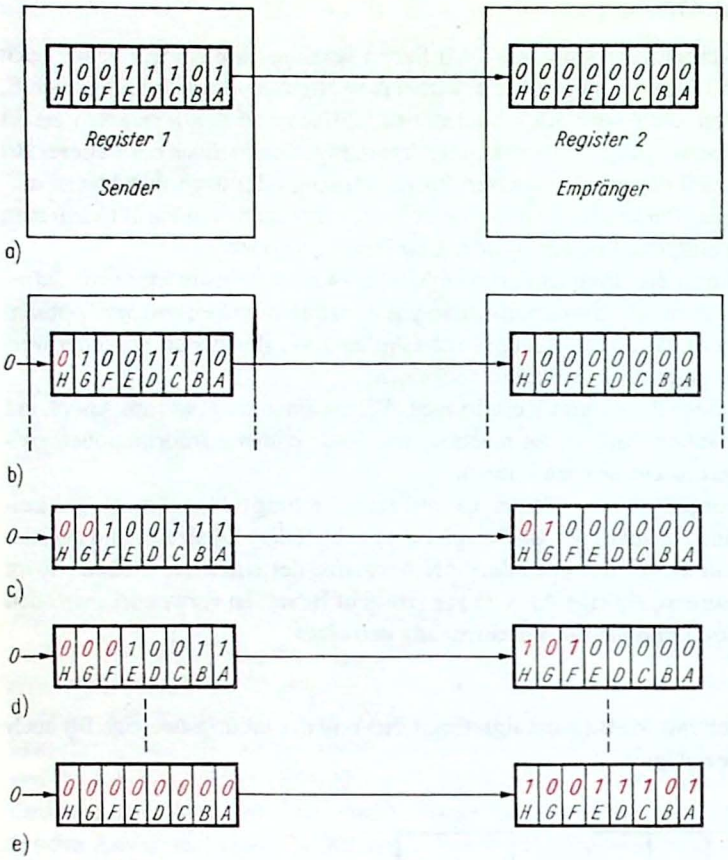


Bild 1.17
Schematische Darstellung der seriellen Übertragung zwischen zwei Registern
a) Ausgangszustand; b)Zustand nach dem ersten Schiebeimpuls (Auslöseimpuls); c) Zustand nach dem zweiten Auslöseimpuls; d) Zustand nach dem dritten Auslöseimpuls; e) Endzustand nach Übertragung aller acht Stellenwerte

Will man erreichen, daß nach den acht Auslöseimpulsen auch das Register 1 noch seine ursprüngliche Information behalten hat, muß man mit einer Rückführung am Register 1 arbeiten. Man muß das am Ausgang abgegebene Bit jeweils wieder am Eingang einspeisen. Nach drei Auslöseimpulsen erhält man dann die im **Bild 1.18** dargestellte Situation.

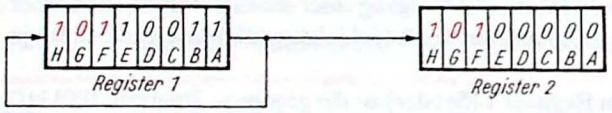


Bild 1.18
Serielle Informationsübertragung zwischen zwei Registern mit Rückgewinnung der Information im Senderegister Reg. 1

Hat man drei dualcodierte Wörter zu übertragen, gibt es die in den **Bildern 1.19a bis c** aufgeführten Übertragungsmöglichkeiten. Die rein serielle Übertragung nach Bild 1.19a erfordert im Beispiel 24 Auslöseimpulse. Die Übertragungsform nach Bild 1.19b erfordert ebenfalls 24 Auslöseimpulse und zusätzlich drei Umschaltimpulse. Sie weist aber Vorteile für den Fall auf, daß wahlweise 1, 2, oder 3 Byte übertragbar sein sollen.
Zeitgünstiger ist eine gleichzeitige Übertragung der drei Wörter über drei Leitungen (gemischte Seriell-Parallel-Übertragung nach Bild 1.19c). Hier wird die gesamte Information der drei Wörter mit *acht* Auslöseimpulsen übertragen.

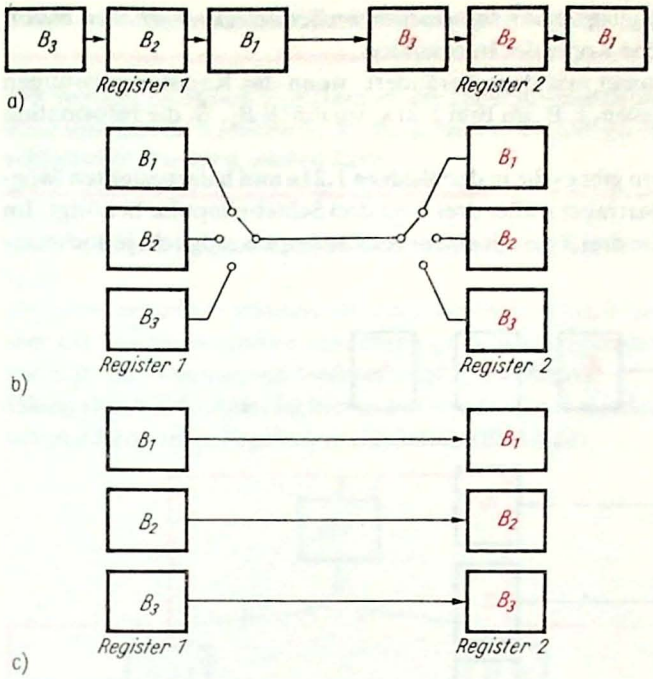


Bild 1.19
Übertragungsmöglichkeiten für drei dualcodierte Wörter

a) rein serielle Übertragung; b) wortweise serielle Übertragung über eine Leitung mit Umschalter; c) wortweise serielle Übertragung bei gleichzeitiger Übertragung der drei Wörter über drei Leitungen
(Die schwarzen Buchstaben kennzeichnen die Ausgangslage, die roten Buchstaben bezeichnen die Lage der einzelnen Wörter nach erfolgter Übertragung.)

Parallelübertragung. Das letzte Beispiel bei der bitseriellen Übertragung (Bild 1.19c) wies schon Merkmale von parallelen Übertragungsanordnungen auf. Als Parallelübertragung bezeichnet man die *gleichzeitige* Übertragung aller acht Bit eines Byte, wie sie im **Bild 1.20** dargestellt ist. Hierbei wird nur *ein* Auslöseimpuls zur gleichzeitigen Übertragung aller acht Bitstellen benötigt. Dafür steigt der Aufwand an Übertragungsleitungen. Für jede Bitstelle ist bei der Parallelübertragung eine Leitung erforderlich.

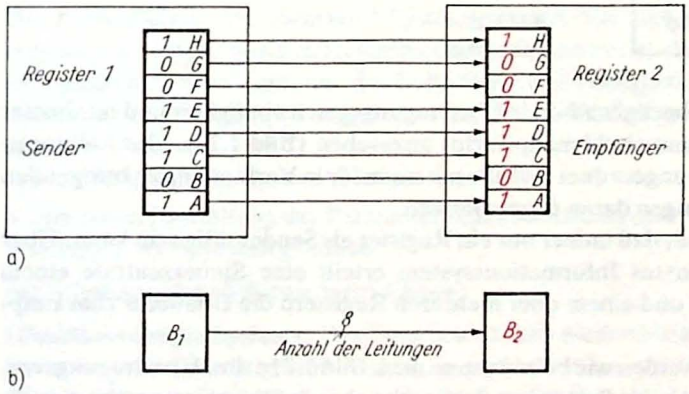


Bild 1.20
Schematische Darstellung der parallelen Informationsübertragung zwischen zwei Registern

a) bei Darstellung aller acht Leitungen; b) bei Zusammenfassung der acht Leitungen zu einem Bündel

Beachten Sie bitte, daß bei dieser Übertragungsart die Information im Senderegister erhalten bleibt. Das Empfangsregister übernimmt nur eine Kopie der Information.

Der Inhalt eines Senderegisters wird immer erst dann verändert, wenn das Register gezwungen wird, eine neue Information zu übernehmen, z. B. im Bild 1.21a, wo durch B_2, B_1 die Information von B_3 geschoben wird.

Zur Parallelübertragung von drei Wörtern gibt es die in den **Bildern 1.21a und b** dargestellten Möglichkeiten. Im Bild 1.21a werden zur Übertragung aller drei Byte drei Schiebeimpulse benötigt. Im Bild 1.21b ist die Parallelübertragung aller drei Byte mit einem Auslöseimpuls möglich, jedoch werden 24 Leitungen benötigt.

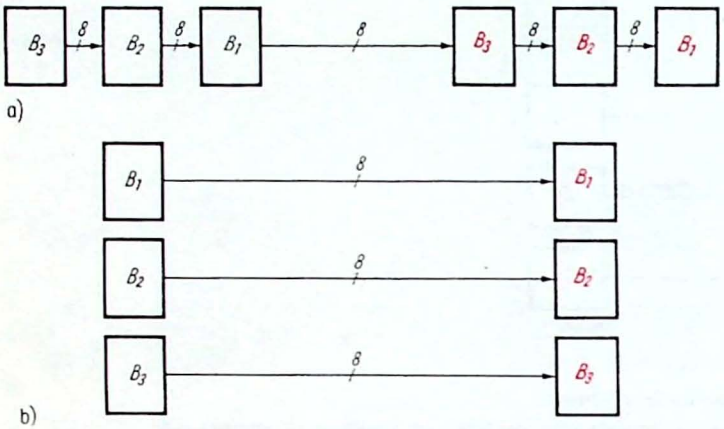


Bild 1.21
Übertragungsmöglichkeiten für drei Wörter in vorwiegend paralleler Art
a) Parallelübertragung in drei seriellen Schüben; b) reine Parallelübertragung unter Verwendung von 24 Leitungen

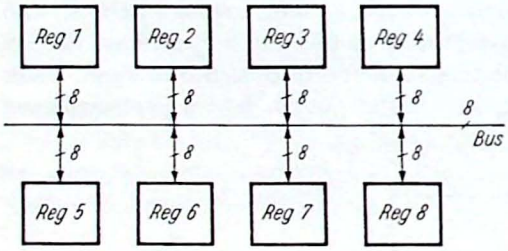


Bild 1.22
Busprinzip

Busprinzip. Als Kompromißlösung bezüglich hoher Übertragungsgeschwindigkeit und minimaler Leitungszahl ist das Busprinzip (Sammelschienenprinzip) anzusehen **(Bild 1.22)**. Die Leitungen des Busses werden räumlich optimal angeordnet und alle miteinander in Verbindung zu bringenden Register mit möglichst kurzen Leitungen daran angeschlossen.

Beim Busprinzip besteht der Nachteil, daß immer nur ein Register als Sender tätig sein kann. Über ein im Bild 1.22 nicht mitgezeichnetes Informationssystem erteilt eine Steuerzentrale einem Baustein die Erlaubnis zum Senden und einem oder mehreren Registern die Erlaubnis zum Empfang.

Bei der Übertragung von drei Byte werden wie beim System nach Bild 1.21a drei Übertragungsvorgänge benötigt. Aus dieser Sicht sind beide Systeme gleichwertig, aber das Busprinzip weist die größere Universalität auf. Beim Bus ist die Reihenfolge der informationsübertragenden Glieder beliebig wählbar.

1.4.5. Zusammenwirken von mehreren informationsverarbeitenden Systemen

Wir wissen: Zwischen Registern ist der Datenaustausch im Serien- oder Parallelbetrieb möglich. Eine universelle Variante ist die Busstruktur, die jedoch den Nachteil hat, das immer nur ein angeschlossener Baustein senden kann.

Nach welchen Regeln können nun mehrere infomationsverarbeitende Systeme zusammenarbeiten?
Unter einem informationsverarbeitenden System (im folgenden kurz I-System genannt) wollen wir hier ein System verstehen, das bereits eine hochorganisierte Form der Informationsverarbeitung, wie z. B. die Analyse von Meßwertfolgen, vornimmt.
Hierarchisches System. Im hierarchischen System hat jedes I-System einem I- System der nächsthöheren Ebene seine Ergebnisse zuzuleiten **(Bild 1.23)**.

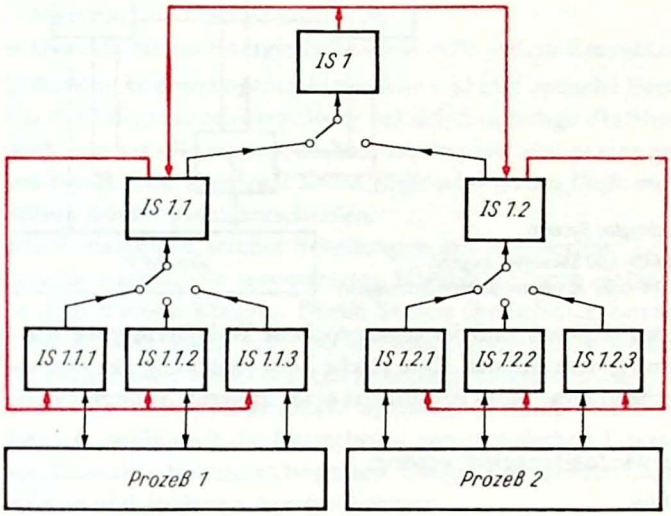


Bild 1.23
Hierarchisches System
(schwarz: Meldeleitungen;
rot: Anweisungsleitungen
von übergeordneten Systemen)

Die I-Systeme der nächsthöheren Ebene fragen die ihnen untergeordneten I-Systeme entsprechend den Erfordernissen ab und verdichten die empfangenen Informationen für die zuständigen I-Systeme der nächsthöheren Ebene.
Die Entscheidung des höchsten I-Systems werden den nächstniederen I-Systemen mitgeteilt und von diesen, entsprechend aufgeschlüsselt, wiederum zur nächsttieferen Ebene weitergeleitet.
Das gesamte System muß, um eine hohe Sicherheit bezüglich Ausfall von Einzelsystemen oder Systemteilen zu erreichen, so aufgebaut sein, daß die Einzelsysteme im Bedarfsfall völlig selbständig arbeiten können.
Jedes übergeordnete System dient also zur

- optimalen Einstellung der Parameter (Eigenschaften) und
- Analyse der Arbeitsergebnisse

der I-Systeme der nächstniederen Ebene.

Gleichberechtigte Systeme. Das Gegenstück zum hierarchischen System ist so konzipiert, daß alle Systeme gleichberechtigt sind. Um in diesem Fall ein koordiniertes Zusammenarbeiten zu gewährleisten, müssen sich die Teilsysteme im Bedarfsfall über die Optimierung des Arbeitsprozesses verständigen können. Jedes I-System muß mit jedem anderen I-System, mit dem es irgendeine Zusammenarbeitsbeziehung hat, in Informationsaustausch treten können **(Bild 1.24)**. Bei räumlich weit

auseinanderliegenden I-Systemen müssen dann Informationsübertragungsstrecken ähnlich unserem Fernsprechsystem aufgebaut werden.
Wenn z. B. das System B festgestellt hat, daß ihm die Arbeitsergebnisse des Systems A uneffektive Belastung bringen, muß in einer vorgeschriebenen Verfahrensweise eine Intervention des Systems B mit einer sachdienlichen Reaktion des Systems A enden. Das kann nur gelingen, wenn die optimalen Zielstellungen allen Teilsystemen bekannt sind und ein vorgegebener *Entscheidungsalgorithmus* die Korrekturmaßnahmen vorschreibt.
Die Intervention eines Systems kann erfolgen über

- Bitte zur Absprache
- Bitte zur Korrektur der Arbeitsweise
- Auslösung eines vorübergehenden Halts des uneffektiven Systems.

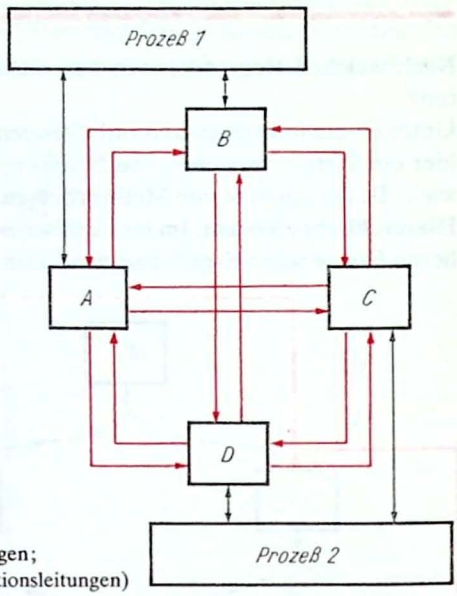


Bild 1.24
Gleichberechtigtes System
(schwarz: Meß- und Steuerleitungen;
rot: Interrupt- oder Kommunikationsleitungen)

Letztlich laufen alle drei Maßnahmen darauf hinaus, daß das angesprochene Teilsystem seine Aufmerksamkeit dem ansprechenden Partnersystem widmet. Eine solche Unterbrechung der eigentlichen Arbeit eines I-Systems zur Entgegennahme eines Einspruchs eines anderen Systems nennt man einen *Interrupt* (s. Abschn. 4.)
Auf die Bitte zur Absprache kann dabei wie folgt reagiert werden:

- Annahme der „Gesprächs"-Anmeldung
- Erfassung der Absicht des ansprechenden Teilsystems
- Überprüfung der Reaktionsmöglichkeiten
- Beantwortung der Anfrage durch ein Angebot von Reaktionsmöglichkeiten
- Warten auf die vom ansprechenden System ausgewählte Variante.

Im Fall einer drohenden Havarie muß eine zwangsweise Unterbrechung mit vorübergehendem Stopp möglich sein. In den anderen Fällen muß dem angesprochenen System ein genügend breiter zeitlicher Spielraum gewährt werden, innerhalb dessen eine Beantwortung der Anfrage erfolgen muß.
Mischsysteme. Die Kombination zwischen hierarchischem und gleichberechtigtem System soll als Mischsystem bezeichnet werden. Diese Form wird sicherlich in vielen Einsatzfällen optimale Bedingungen schaffen. Trotzdem soll hier nicht weiter darauf eingegangen werden, da man sich beliebig viele und beliebig komplizierte Varianten ausdenken kann.
Ein Entwickler solcher Systeme steht vor der Aufgabe, ein Vorschriftenwerk für die Zusammenarbeit aller Teilsysteme zu entwerfen, das eine widerspruchsfreie und optimal effektive Zusammenarbeit garantiert. Je komplizierter die Struktur solcher Systeme gestaltet wird, um so mehr Änlichkeit wird man zu den Systemen des menschlichen Zusammenwirkens entdecken. Dabei zeigt sich beim derzeitigen Stand der Automaten, daß sie die eingegebenen Arbeitsvorschriften (Programme) bei fehlerfreiem Betrieb kompromißlos einhalten.

2. Informationsverarbeitung durch einfache Logiksysteme

Nachdem wir im Abschnitt 1. grundlegende Einsichten in die Methoden der Informationsverarbeitung erworben haben, wollen wir uns mit der Frage befassen, wie man technische Einrichtungen gestalten muß, damit sie Informationen automatisch verarbeiten können. Wir werden dabei mit den einfachsten Gebilden beginnen und uns allmählich bis zum Mikrorechner vorarbeiten.
Wir wissen bereits, warum man sich bevorzugt der elektrischen Energie bei der automatischen Informationsverarbeitung bedient. Die Gründe sind:

- Plötzliche Zustandsänderungen der elektrischen Spannungen oder der elektrischen Stromstärke werden mit hoher Geschwindigkeit übertragen.
- Stromstärke und Spannung sind durch elektronische Schaltungen steuerbar, die sehr wenig Material und Energie erfordern.
- Die elektrische Energie läßt sich in viele andere Energiearten umformen.

Zukünftig könnten optoelektronische und rein optische Bauelemente Bedeutung erlangen.
Für die Informationsdarstellung hat sich heutzutage das Muster aus Binärzeichen eingebürgert. Es wird, wie wir alle wissen, deshalb verwendet, weil es eine große Sicherheit gegenüber Störeinflüssen bietet. Die Zustände *Strom fließt* oder *Strom fließt nicht* lassen sich auch noch bei stärkeren Störeinflüssen gut unterscheiden.
Die Entwicklung solcher Schaltungen mit zweiwertiger Logik begann mit dem Einsatz einfacher Schaltkontakte, die entweder von Hand oder durch elektromagnetische Kraftwirkungen (Relais) betätigt werden können. Durch Stellen der Schalterkontakte von Hand kann der Mensch seine *Steuerabsicht* realisieren. Bei der Betätigung der Schaltkontakte durch Elektromagnete ist die Möglichkeit der *automatischen Informationsverarbeitung* gegeben.
Obwohl heute Relaiskontakte immer mehr durch vollelektronische Logikschaltungen verdrängt werden, wollen wir die Betrachtung von technischen Informationsverarbeitungssystemen mit der von Kontaktschaltungen beginnen. Diese haben für den Lernenden den Vorteil der leichteren Faßlichkeit und größeren Anschaulichkeit.

2.1. Einfache elektrische Logiksysteme ohne Rückführung

Diese Systeme übertragen den Informationsfluß nur in einer Richtung vom Eingang zum Ausgang. Das Ausgangssignal wirkt nicht auf den Eingang oder auf andere Punkte des Übertragungswegs zurück. Sie werden auch als kombinatorische Schaltungen bezeichnet.

2.1.1. Elementare Logikschaltungen

2.1.1.1. UND-Funktion

Zur Veranschaulichung der UND-Funktion wollen wir mit einer einfachen Aufgabenstellung beginnen.
Es soll eine Teilschaltung für eine Aufzugssteuerung angegeben werden, die den Aufzug nur dann anfahren läßt, wenn
a) die Aufzugschachttür geschlossen ist

b) eine Taste vom Aufzugsbenutzer gedrückt worden ist, die mit ihrem Schaltzustand die Aufzugsteuerung über das zu erreichende Stockwerk informiert.

Zur praktischen Lösung dieser Aufgabe **(Bild 2.1)** soll ein Türkontakt S1 verwendet werden, der dann geschlossen ist, wenn auch die Aufzugschachttür geschlossen ist. Beim Drücken einer beliebigen Stockwerksanwahltaste soll gleichzeitig der Schalter S2 geschlossen werden. Ordnet man diese zwei Schalterkontakte in einer *Reihenschaltung* an, so kann der Strom in diesem Kreis erst dann fließen, wenn alle beide Kontakte geschlossen sind. Der Stromfluß im Kreis führt zum Anziehen des Relais Rs, wodurch weitere Steuermaßnahmen eingeleitet werden, die letztlich zum Anlaufen des Motors führen.

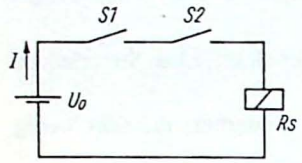


Bild 2.1
Reihenschaltung von zwei Schalterkontakten

Dieser offensichtliche und einfache Sachverhalt läßt sich in einem kurzen Aussagesatz darstellen:

- In der Schaltung nach Bild 2.1 fließt Strom, wenn die Kontakte S1 und S2 geschlossen sind. (1)

Für die in diesem Aussagesatz auftretenden Zustände wollen wir ein Zustandssystem nach **Tafel 2.1** definieren, in dem für jede definierte Variable nur zwei mögliche Werte 0 oder 1 zugelassen werden.

Tafel 2.1. Zustandssystem für die Schaltung nach Bild 2.1

Physikalischer Zustand	Symbolische Zustandskennzeichnung
Strom *I* fließt	$y = 1$
Strom *I* fließt nicht	$y = 0$
Kontakt S 1 geschlossen	$x_1 = 1$
Kontakt S 1 nicht geschlossen	$x_1 = 0$
Kontakt S 2 geschlossen	$x_2 = 1$
Kontakt S 2 nicht geschlossen	$x_2 = 0$

Vorher wollen wir noch vereinbaren:

- Kontakte oder Anschlußklemmen werden mit großen Buchstaben bezeichnet. *Variable*, die Zustände (z. B. von Schaltkontakten oder von Spannungen oder Stromstärken) beschreiben, werden durch kleine Buchstaben abgekürzt dargestellt.

Tafel 2.1 bietet den Vorteil, daß wir alle Formulierungen kürzer fassen können. So kann man z. B. die Aussage *Strom fließt* ersetzen durch $y = 1$ oder *Kontakt* S1 *nicht geschlossen* ersetzen durch $x_1 = 0$.

Den gesamten Aussagesatz (1) kann man nun ebenfalls wesentlich kürzer formulieren, indem man einen besonders definierten algebraischen Ausdruck verwendet. Dafür sind folgende Formen möglich:

$y = x_1 \wedge x_2$ Form 1

$y = x_1 \,\&\, x_2$ Form 2

$y = x_1 \cdot x_2 = x_1 x_2$ Form 3.

Alle drei Formen beinhalten nach Definition die gleiche Aussage. Die ersten beiden führen bei umfangreichen Ausdrücken auf ein kompliziert zu lesendes Schriftbild; die dritte Form ist am einfachsten und sagt uns sicherlich am meisten zu. Sie hat nur den Nachteil, daß man sie mit einer Gleichung zur Multiplikation der zwei Variablen x_1, x_2 verwechseln kann. Wir wollen sie trotzdem benutzen, da sie sich z. Z. in Technikerkreisen durchgesetzt hat und sogar in TGL 16056 verwendet wird.

Die Gleichung $y = x_1 x_2$ beinhaltet folgende allgemeine Aussage:

- y nimmt dann den Zustandswert 1 an, wenn sowohl $x_1 = 1$ und auch $x_2 = 1$ sind.

Wegen der sprachlichen *und*-Verbindung wird diese Art der Variablenverknüpfung UND-Funktion genannt.

- Das mathematische Spezialgebiet, in dem logische Zustandsverknüpfungen von binären Variablen behandelt werden, wird Boolesche Algebra (nach dem Schöpfer dieser Theorie) oder auch Schaltalgebra genannt.

Wir haben nun, ausgehend von einer Reihenschaltung zweier Schalterkontakte, ein Zustandssystem (Tafel 2.1), einen Aussagesatz (1) sowie eine algebraische Kurzschriftform für diesen Aussagesatz formuliert bzw. definiert. Wir sind von einem konkreten Sachverhalt zu einer mathematischen Darstellung übergegangen, indem wir von dem konkreten Sachverhalt nur das uns Interessierende, das Wesentliche, beschrieben haben. Einen solchen Prozeß nennt man eine *Abstraktion*.

Der Vorteil dieser Abstraktion besteht darin, daß wir anstelle der Reihenschaltung zweier Kontakte mit einem mathematischen Modell, der UND-Funktion, weiterarbeiten können. **Bild 2.2** zeigt uns noch einmal diesen Prozeß in anschaulicher Form.

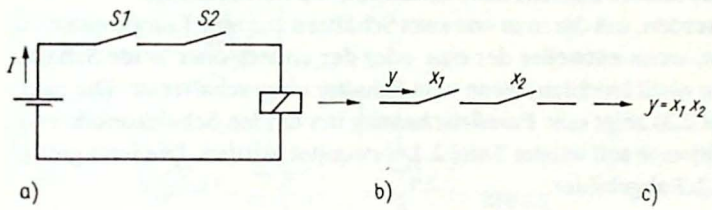


Bild 2.2
Übergang von der Reihenschaltung zweier Kontakte zur UND-Funktion
a) vollständige UND-Schaltung; b) Abstraktion; c) UND-Funktion

Man kann dabei einen Schritt weitergehen und an die Kontakte im Bild 2.2b die Variablen x_1, x_2 schreiben. Damit soll darauf hingewiesen werden, daß die Zustände der Schalterkontakte durch diese Variablen ausgedrückt werden. Wir wollen aber keinesfalls unsere vorher getroffene Vereinbarung brechen und plötzlich die Schalterkontakte mit kleinen Buchstaben bezeichnen. Es bleibt dabei, daß der konkrete Kontakt mit einem Großbuchstaben (hier S) und seine Zustandsvariable mit einem kleinen Buchstaben (hier x) beschrieben werden.

Tafel 2.2
Schaltbelegungstabelle für eine UND-Funktion

Nr.	x_2	x_2	$y = x_1 x_2$
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

Abschließend müssen wir noch einen Sachverhalt besprechen. Jede der Variablen x_1, x_2 kann bekanntlich zwei Zustände einnehmen. Das bedeutet in diesem Fall, daß insgesamt vier verschiedene Zustandskombinationen von x_1, x_2 auftreten können. Ordnet man die möglichen Wertekombinationen von x_1, x_2 nach fortschreitenden Dualzahlen, so erhält man die in **Tafel 2.2** abgebil-

dete Übersicht. In dieser Kombinationstabelle, die allgemein unter dem Namen *Schaltbelegungstabelle* bekannt ist, kann aus jeder Zeile eine bestimmte Aussage abgelesen werden. Bezieht man die Aussagen auf unser Beispiel, können folgende Sätze formuliert werden:

Zeile Nr. 0 Sind beide Schalter S1 und S2 geöffnet ($x_1 = 0, x_2 = 0$), so fließt kein Strom ($y = 0$).
Zeile Nr. 1 Ist S1 geschlossen ($x_1 = 1$) und ist S2 geöffnet ($x_2 = 0$), so fließt kein Strom ($y = 0$)
usw.

In mathematischer Kurzform lauten die Aussagen der vier Zeilen:

allgemein $x_2 \cdot x_1 = y$

Zeile Nr. 0 $0 \cdot 0 = 0$

Zeile Nr. 1 $0 \cdot 1 = 0$

Zeile Nr. 2 $1 \cdot 0 = 0$

Zeile Nr. 3 $1 \cdot 1 = 1.$

Die Schaltbelegungstabelle stellt also eine Tabelle der *möglichen* Wertekombinationen und der damit erzielbaren logischen Aussagen dar. Im Grunde liegt hier der gleiche Sachverhalt vor wie in der allgemeinen Algebra. Zu jeder Gleichung oder Funktion kann man eine Wertetabelle anfertigen. Im Gegensatz zur allgemeinen Algebra haben jedoch die Funktionen der Schaltalgebra eine meist überschaubare Wertemannigfaltigkeit.

2.1.1.2. ODER-Funktion

Das nächste Beispiel soll uns an eine andere logische Elementarfunktion heranführen.
Es soll eine Schaltung angegeben werden, mit der man von zwei Schaltern aus eine Lampe einschalten kann. Die Lampe soll leuchten, wenn entweder der eine oder der andere oder beide Schalter eingeschaltet sind. Sie soll nur dann nicht leuchten, wenn kein Schalter eingeschaltet ist. Die praktische Lösung dieser Aufgabe **(Bild 2.3)** zeigt eine *Parallelschaltung* der beiden Schaltkontakte S1, S2. Als Tabelle der Zustandsdefinitionen soll wieder Tafel 2.1 verwendet werden. Die jetzt gültige Schaltbelegungstabelle ist in **Tafel 2.3** abgebildet.

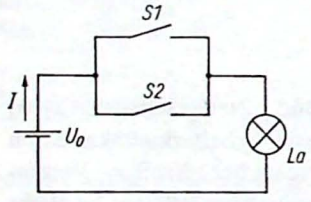


Bild 2.3. Parallelschaltung von zwei Schalterkontakten

Tafel 2.3. Schaltbelegungstabelle für eine ODER-Funktion

x_2	x_1	$y = x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

Aus der Aufgabenstellung und der Schaltung im Bild 2.3 kann der folgende Aussagesatz abgeleitet werden:

- In der Schaltung nach Bild 2.3 fließt Strom, wenn der eine oder der andere oder alle beide Schalter geschlossen sind. (2)

Satz (2) kann mit Hilfe der Zustandsvariablen x_1, x_2 in folgenden schaltalgebraischen Kurzformen dargestellt werden:

$$y = x_1 \vee x_2$$

oder

$$y = x_1 + x_2.$$

Man nennt diese, der sprachlichen *oder*-Verknüpfung entsprechende Aussage eine ODER-Funktion (Inclusiv-ODER, einschließendes ODER).

Obwohl uns hier die Schreibweise mit dem Pluszeichen am meisten zusagen wird, wollen wir doch mit dem ∨-Zeichen arbeiten, damit keine Verwechslungen mit der Addition auftreten können. Für die ODER-Funktion mit zwei Variablen erhält man die im **Bild 2.4** gezeigte abstrakte Gleichwertigkeitsdarstellung. Tafel 2.3 liefert aus der allgemeinen Beziehung $y = x_1 \vee x_2$ folgende Einzelaussagen:

$$0 \vee 0 = 0 \qquad 1 \vee 0 = 1$$
$$0 \vee 1 = 1 \qquad 1 \vee 1 = 1.$$

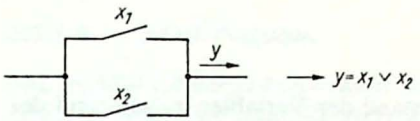


Bild 2.4
Übergang von der Parallelschaltung zweier Kontakte zur ODER-Funktion

2.1.1.3. Negation

Mit Hilfe der im **Bild 2.5** dargestellten Schaltung, bei der ein Relais Rs die Kontakte K1 und K2 betätigt, soll die dritte elementare Logikfunktion eingeführt werden. Zur Zustandsdefinition wollen wir **Tafel 2.4** verwenden.

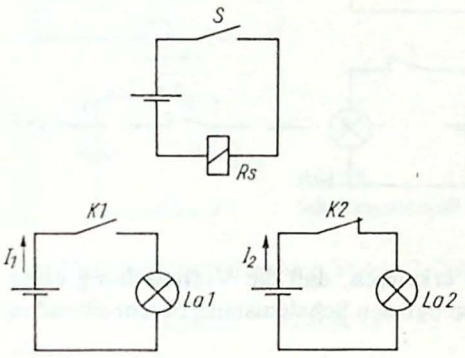


Bild 2.5
Relaisschaltung zur Erläuterung der Negation

Tafel 2.4. Zustandssystem für die Schaltung nach Bild 2.5

Physikalischer Zustand Urache	Folgereaktion	Symbolische Zustandskennzeichnung
Strom I_1 fließt Strom I_1 fließt nicht	Lampe La 1 leuchtet Lampe La 1 leuchtet nicht	$y_1 = 1$ $y_1 = 0$
Strom I_2 fließt Strom I_2 fließt nicht	Lampe La 2 leuchtet Lampe La 2 leuchtet nicht	$y_2 = 1$ $y_2 = 0$
Kontakt S geschlossen	Kontakt K 1 geschlossen Kontakt K 2 geöffnet	$x = 1$
Kontakt S geöffnet	Kontakt K 1 geöffnet Kontakt K 2 geschlossen	$x = 0$

Wir gewinnen aus der Schaltung nach Bild 2.5 und mit Hilfe des Zustandssystems nach Tafel 2.4 folgende Aussagen:

- Ist Schalter S geöffnet ($x = 0$), fließt Strom I_1 nicht ($y_1 = 0$), und Strom I_2 fließt ($y_2 = 1$).
- Ist Schalter S geschlossen ($x = 1$), fließt Strom I_1 ($y_1 = 1$), und Strom I_2 fließt nicht ($y_2 = 0$).

Daraus läßt sich folgern, daß der Strom I_1 nur dann fließt, wenn der Strom I_2 nicht fließt und umgekehrt. Die Zustände dieser beiden Ströme sind also immer entgegengesetzt. Man sagt, daß der Zustand des einen Stromes die Negation des Zustands des anderen Stromes darstellt und drückt die Negation in der schaltalgebraischen Schreibweise durch einen Querstrich über der Variablen aus. **Tafel 2.5** zeigt eine Zusammenstellung der möglichen Zustände der Schaltung im Bild 2.5. Aus dieser Tafel leiten sich die folgenden allgemeinen Schreibweisen ab:

$$y_1 = x$$

$$y_2 = \bar{x} \qquad \text{(gelesen: } y_2 \text{ ist gleich } x \text{ negiert)},$$

folglich

$$y_1 = \bar{y}_2 \quad \text{und} \quad y_2 = \bar{y}_1.$$

Der Zustand der Variablen y_1 entspricht also stets dem Zustand der Variablen x, während der Zustand der Variablen y_2 stets dem negierten Zustand von x gleich ist. Mit diesen Aussagen und Tafel 2.5 folgt auch

$$\bar{0} = 1$$

$$\bar{1} = 0.$$

x	y_1	y_2
0	0	1
1	1	0

Tafel 2.5. Schaltbelegungstabelle für die Negation $y_2 = \bar{y}_1 = \bar{x}$

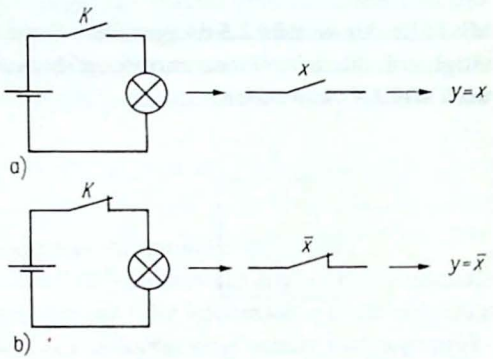


Bild 2.6
Übergang von der Kontaktdarstellung zur Logikfunktion
a) Identität (Gleichheit); b) Negation

Der Abstraktionsprozeß ist im **Bild 2.6** dargestellt. Sie erkennen, daß die Vertauschung eines Arbeitskontakts durch einen Ruhekontakt einer Negation der den Schaltzustand beschreibenden Variablen entspricht.

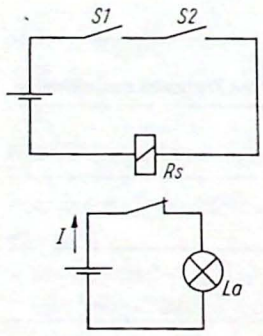


Bild 2.7
Schaltungsbeispiel zur NAND-Funktion

Tafel 2.6. Schaltbelegungstabelle für eine NAND-Funktion

x_2	x_1	$y = \overline{x_1 x_2}$
0	0	1
0	1	1
1	0	1
1	1	0

2.1.1.4. NAND-Funktion

Die Schaltung nach **Bild 2.7** bringt uns auf die NAND-Funktion. Nehmen wir wieder die Tabelle der Zustandsdefinitionen nach Tafel 2.1, so erhalten wir die Schaltbelegungstabelle nach **Tafel 2.6.** Vergleichen wir Tafel 2.2 mit Tafel 2.6, erkennen wir, daß die y-Werte der einen Tafel den jeweils negierten Werten der anderen Tafel entsprechen. Das kann doch nur bedeuten, daß die zu den Ta-

feln 2.2 und 2.6 gehörenden schaltalgebraischen Funktionen durch Negation auseinander hervorgehen. Damit lassen sich folgende Gleichungen formulieren:

$$y_{\text{TAFEL2.6}} = \bar{y}_{\text{TAFEL2.2}} = \overline{x_1 x_2}$$

oder

$$y_{\text{TAFEL2.6}} = \overline{x_1 x_2} \qquad \text{(NAND-Funktion).}$$

Damit haben wir die schaltalgebraische Formulierung der NAND-Funktion gefunden. Der Ausdruck NAND leitet sich aus NOT AND (NICHT UND), also dem negierten UND, ab.

2.1.1.5. NOR-Funktion

Die im **Bild 2.8** dargestellte Schaltung führt auf die NOR-Funktion. Die Bezeichnung NOR leitet sich von NOT OR (NICHT ODER) ab, entspricht folglich dem negierten ODER. Mit den Zustandsdefinitionen nach Tafel 2.1 erhalten wir die Schaltbelegungstabelle nach **Tafel 2.7**.

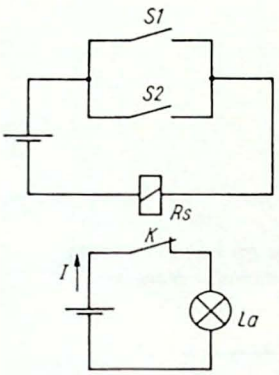


Bild 2.8
Schaltungsbeispiel zur NOR-Funktion

Tafel 2.7. Schaltbelegungstabelle für eine NOR-Funktion

x_2	x_1	$y = \overline{x_1 \vee x_2}$
0	0	1
0	1	0
1	0	0
1	1	0

Der Vergleich zwischen den Tafeln 2.7 und 2.3 ergibt

$$y_{\text{TAFEL 2.7}} = \bar{y}_{\text{TAFEL 2.3}} = \overline{x_1 \vee x_2}$$

also

$$y_{\text{TAFEL 2.7}} = \overline{x_1 \vee x_2} \qquad \text{(NOR-Funktion).}$$

2.1.2. Theoreme von *de Morgan*

Die Schaltung nach **Bild 2.9** führt uns auf einen wichtigen Zusammenhang. Hierzu legen wir die Tabelle der Zustandsdefinition nach **Tafel 2.8** fest und erhalten dann die in **Tafel 2.9** dargestellte Schaltbelegungstabelle.
Durch Vergleich der Tafeln 2.9 und 2.7 erhält man

$$\bar{x}_1 \bar{x}_2 = \overline{x_1 \vee x_2}. \qquad (a_1)$$

Negiert man beide Seiten der Gleichung, so folgt

$$\overline{\bar{x}_1 \bar{x}_2} = \overline{\overline{x_1 \vee x_2}}$$

oder (unter Beachtung, daß sich eine doppelte Negation aufhebt)

$$\overline{\bar{x}_1 \bar{x}_2} = x_1 \vee x_2. \qquad (a_2)$$

Der Vergleich der Tafeln 2.9 und 2.6 ergibt

$$\bar{x}_1 \vee \bar{x}_2 = \overline{x_1 x_2} \qquad (b_1)$$

und durch Negation beider Gleichungsseiten

$$\overline{\bar{x}_1 \vee \bar{x}_2} = x_1 x_2. \qquad (b_2)$$

Die Gln. (a_1), (a_2) stellen das eine Theorem, die Gln. (b_1), (b_2) das andere Theorem von *de Morgan* dar.

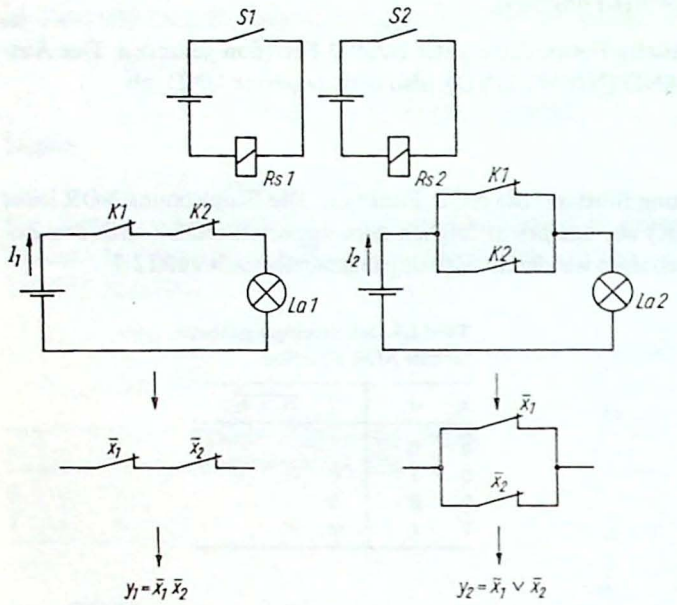


Bild 2.9
Schaltungsbeispiel zur Erläuterung der Theoreme von *de Morgan*

Tafel 2.8. Zustandssystem für die Schaltung nach Bild 2.9

Physikalischer Zustand		Symbolische Zustandskennzeichnung
Ursache	Folgereaktion	
Strom I_i fließt	Lampe La_i leuchtet	$y_i = 1$
Strom I_i fließt nicht	Lampe La_i leuchtet nicht	$y_i = 0$
Kontakt S_i geschlossen	Kontakt K_i geöffnet	$x_i = 1$
Kontakt S_i geöffnet	Kontakt K_i geschlossen	$x_i = 0$

x_2	x_1	$y_1 = \bar{x}_1 \bar{x}_2$	$y_2 = \bar{x}_1 \vee \bar{x}_2$
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

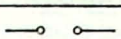
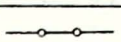
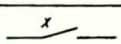
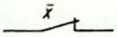
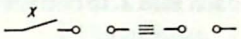
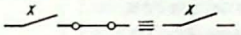
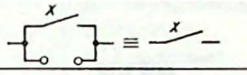
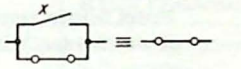
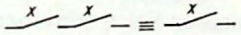
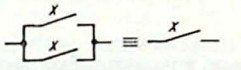
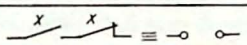
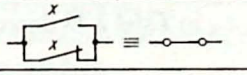
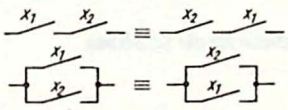
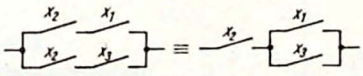
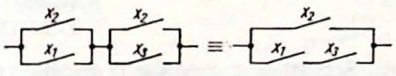
Tafel 2.9
Schaltbelegungstabelle für die Schaltung nach Bild 2.9

2.1.3. Regeln und Anwendung der Schaltalgebra

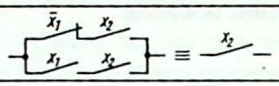
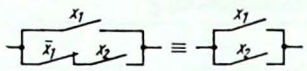
Ehe wir nun zwei Beispiele von kombinierten Schaltungen betrachten, wollen wir noch einige Regeln der Schaltalgebra kennenlernen. Wir sehen uns hierzu Tafel 2.10 an. Zuvor wollen wir vereinbaren, daß eine Leitungsunterbrechung mit dem Zustand 0, eine ständig überbrückte Leitungsöffnung mit dem Zustand 1 bezeichnet werden sollen. Außerdem können Sie über die Gleichwertigkeit der ermittelten Kontaktschaltungen in den Regeln 14 bis 17 erkennen, daß Ausklammern und Ausmultiplizieren wie in der allgemeinen Algebra durchgeführt werden können, wenn man das Operationszeichen $\vee$ wie das Pluszeichen bei der Addition wertet.

Da die **Tafel 2.10** Veranschaulichungen und Beweisführungen enthält, sollen keine weiteren Erläuterungen gegeben werden. Sie erarbeiten sich diesen Tafelinhalt am günstigsten, wenn Sie nach Möglichkeit die aufgeführten Regeln selbst beweisen.

Tafel 2.10. Einige Regeln der Schaltalgebra

Nr.	Regel	Veranschaulichung	Beweis
1	$y = 0$		ständig offene Leitung
2	$y = 1$		ständig geschlossener Kontakt
3	$y = x$		Arbeitskontakt. Der Zustandswert von y ist mit dem Zustandswert von x identisch
4	$y = \bar{x}$		Ruhekontakt. Der Zustandswert von y ist gleich der Negation des Zustandswerts von x
5	$y = x \cdot 0 = 0$		Der Zustand von x übt keinen Einfluß auf y aus
6	$y = x \cdot 1 = x$		Nur der Zustand von x wirkt
7	$y = x \vee 0 = x$		Das offene Leitungsstück ist wirkungslos
8	$y = x \vee 1 = 1$		Der ständig geschlossene Kontakt setzt sich durch (ODER-Funktion)
9	$y = x \cdot x = x$		Der Zustand beider Kontakte ist stets gleich. Beide sind entweder offen oder geschlossen
10	$y = x \vee x = x$		
11	$y = x \cdot \bar{x} = 0$		Ein Kontakt ist stets offen
12	$y = x \vee \bar{x} = 1$		Ein Kontakt ist stets geschlossen
13	$y = x_1x_2 = x_2x_1$ $y = x_1 \vee x_2 = x_2 \vee x_1$		Die Reihenfolge der Kontakte ist gleichgültig. Die Variablen können in der Gleichung ausgetauscht werden
14	$y = x_2x_1 \vee x_2x_3$ $= x_2 (x_1 \vee x_3)$		Soll Strom fließen, muß x_2 in jedem Fall geschlossen sein (Ausklammerregel)
15	$y = (x_2 \vee x_1)\ (x_2 \vee x_3)$ $= x_2 \vee x_1 x_3$		Anwendung des Ausmultiplizierens $(x_2 \vee x_1)\ (x_2 \vee x_3)$ $= x_2x_2$ $\vee\ x_2x_3 \vee x_1x_2 \vee x_1x_3$ $= x_2 \vee x_2 (x_1 \vee x_3) \vee x_1x_3$ $= x_2 (1 \vee x_1 \vee x_3) \vee x_1x_3$ $= x_2 \vee x_1x_3$

Tafel 2.10. (Fortsetzung)

Nr.	Regel	Veranschaulichung	Beweis
16	$y = \bar{x}_1 x_2 \vee x_1 x_2 = x_2$		$\bar{x}_1 x_2 \vee x_1 x_2$ $= x_2 (x_1 \vee \bar{x}_1) = x_2$
17	$y = x_1 \vee \bar{x}_1 x_2 = x_1 \vee x_2$		Erweitern mit $x_1 \vee \bar{x}_1 = 1$. $x_1 \vee x_2$ $= (x_1 \vee x_2)(x_1 \vee \bar{x}_1)$ Nach Ausmultiplizieren und Zusammenfassen wie bei Nr. 15 erhält man: $x_1 \vee x_2 = x_1 \vee \bar{x}_1 x_2$
18	$y = \overline{x_1 \vee x_2} = \bar{x}_1 \bar{x}_2$ $y = \overline{x_1 x_2} = \bar{x}_1 \vee \bar{x}_2$	Theoreme von *de Morgan* (s. Abschn. 2.1.2.)	

Nach dem Studium der Tafel 2.10 wollen wir das erste Beispiel einer aus elementaren Logikschaltungen zusammengesetzten Kombination nach **Bild 2.10** betrachten. Wir können folgende Aussage formulieren:

- In der Schaltung nach Bild 2.10 fließt Strom, wenn S3 eingeschaltet ist und zusätzlich entweder S1 *oder* S2 geschlossen ist.

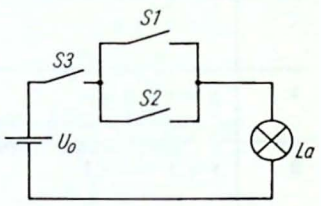


Bild 2.10
Erstes Beispiel einer Kombinationsschaltung

Benutzen wir wieder die Zustandsdefinitionen nach Tafel 2.1, können wir diese Aussage durch die Variablen y, x_1, x_2, x_3 in folgender algebraischen Form schreiben:

$$y = x_3 x_1 \vee x_3 x_2.$$

Mit Hilfe der Regel 14 (Tafel 2.10) kann man zusammenfassend die Form gewinnen:

$$y = x_3 (x_1 \vee x_2). \qquad \text{(a)}$$

Aus dieser logischen Funktion kann man die in **Tafel 2.11** dargestellte Schaltbelegungstabelle ableiten.

Tafel 2.11
Schaltbelegungstabelle für die Schaltung nach Bild 2.10

x_3	x_2	x_1	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Man kann aber auch zuerst die Schaltbelegungstabelle angeben und mit ihrer Hilfe die algebraische Funktion gewinnen. Dazu geht man wie folgt vor:

- Alle Zeilen der Schaltbelegungstabelle, bei denen $y = 1$ ist, werden als UND-Funktionen aufgeschrieben und durch ODER-Verbindungen verknüpft. Hierbei werden die Zustände $x_i = 1$ durch x_i und $x_i = 0$ und durch $\bar{x}_i$ ausgedrückt.

Aus Tafel 2.11 gewinnt man dadurch folgende Aussage:

$$y = x_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee x_1x_2x_3. \qquad \text{(b)}$$

Die Variable y ist also eins, wenn eine der drei Teil-UND-Funktionen eins ergibt.
Dieser Ausdruck läßt sich zu dem bisher ermittelten Ausdruck (a) vereinfachen. Dazu wendet man die bisher bekannten Regeln einschließlich der Ausklammerregel an oder benutzt spezielle Minimierungsverfahren, die wir hier aber nicht besprechen wollen. Durch Ausklammern erhält man aus dem Ausdruck (b):

$$y = x_3\,(x_1\bar{x}_2 \vee \bar{x}_1x_2 \vee x_1x_2)$$

$$y = x_3\,(x_1\bar{x}_2 \vee x_2\,(x_1 \vee \bar{x}_1))$$

$$y = x_3\,(x_1\bar{x}_2 \vee x_2)$$

$$y = x_3\,(x_1 \vee x_2) \qquad \text{(nach Regel 17, Tafel 2.10).}$$

Tafel 2.12. Zustandssystem für die Schaltung nach Bild 2.11

Physikalischer Zustand Ursache	Folgereaktion	Symbolische Zustandskennzeichnung
Strom im Lampenkreis i fließt	Lampe i leuchtet	$y_i = 1$
Strom im Lampenkreis i fließt nicht	Lampe i leuchtet nicht	$y_i = 0$
S1 geschlossen	K11, K13 geschlossen, K12 geöffnet	$x_1 = 1$
S1 geöffnet	K12 geschlossen, K11, K13 geöffnet	$x_1 = 0$
S2 geschlossen	K22, K23 geschlossen, K21 geöffnet	$x_2 = 1$
S2 geöffnet	K21 geschlossen, K22, K23 geöffnet	$x_2 = 0$

Im **Bild 2.11** ist das zweite Beispiel einer kombinierten Schaltung dargestellt, das wir hier besprechen wollen. Dazu gehören die Definitionstabelle nach **Tafel 2.12** und die Schaltbelegungstabelle nach **Tafel 2.13**. Diese Schaltung bewirkt, daß man durch Kombination der Zustände der Schalterkontakte S1 und S2 jeweils nur eine bestimmte Lampe oder gar keine Lampe zum Leuchten bringt. Die Auswahl der Lampen entsprechend den Zuständen der Kontakte S1 und S2 zeigt die Tafel 2.13. Die logischen Funktionen sind sehr einfach:

$$y_1 = x_1\bar{x}_2$$

$$y_2 = \bar{x}_1x_2$$

$$y_3 = x_1x_2.$$

x_2	x_1	y_1	y_2	y_3
0	0	0	0	0
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1

Tafel 2.13
Schaltbelegungstabelle für die Schaltung nach Bild 2.11

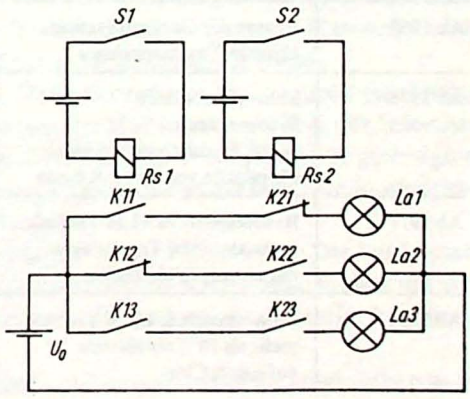


Bild 2.11
Zweites Beispiel einer Kombinationsschaltung

Mit dieser Schaltung haben wir ein bedeutungsvolles Schaltungsprinzip für die Informationsverarbeitung kennengelernt. Die Zustände der Schalterkontakte S1 und S2 kann man als *Adreßcodierung* deuten; denn jede Kombination der Zustandsvariablen x_1, x_2 entspricht einer bestimmten Adresse. Es leuchtet jeweils die Lampe, deren Adresse gebildet wird.
Mit den beiden Kontakten S1 und S2 gelingt es, drei Lampen oder, wenn man die Funktion $y = \bar{x}_1\bar{x}_2$ ebenfalls als Adresse ausnutzt, vier Lampen zu adressieren. Bei drei Kontakten werden $2^3 = 8$ Lampen, bei vier Kontakten $2^4 = 16$ Lampen usw. adressierbar.

2.1.4. Einfluß der Halbleitertechnik

Man könnte nun die Betrachtung von Kontaktschaltungen fortsetzen und zu umfangreichen Relaissystemen übergehen; man könnte mit ihnen sogar einen Digitalrechner entwerfen.
Die technische Entwicklung vollzog sich tatsächlich in dieser Weise. Der erste funktionsfähige programmierbare Digitalrechner wurde 1937 vom deutschen Ingenieur *Conrad Zuse* fertiggestellt. Er verwendete dazu Relais und einen besonders konstruierten mechanischen Speicher. Noch bis 1958 wurden Rechner auf Relaisbasis weiterentwickelt.

Tafel 2.14. Zur Entwicklung der Rechentechnik

Zeitangabe	Bauelemente Baugruppen	EDVA	Volumen	Leistung
Vor 1948	Relais Hochvakuumröhren 1948 wurde der Transistor erfunden	erste elektronische Rechner auf Relais- und Röhrenbasis ENIAC: 18000 Röhren, 200 kW, kleiner Saal	V_0 kleiner Saal	P_0 200 kW
1950–1960	Entwicklung der Transistortechnik in diskreten Schaltungen Einführung der gedruckten Schaltung	EDVA auf Transistorbasis Bei gleicher verarbeitbarer Informationsmenge sinkt nach grober Schätzung der Raumbedarf auf etwa $0{,}1 \cdot V_0$. Ebenso sinkt der Leistungsbedarf auf etwa $0{,}1 \cdot P_0$	$\sim \frac{V_0}{10}$ großes Zimmer	$\sim \frac{P_0}{10}$ 20 kW
1960–1968	Einführung von integrierten Schaltkreisen (Gatter, Trigger, Verstärker) Dieser Trend zur Integration von Funktionsgruppen setzt sich immer stärker fort	EDVA mit integrierten Schaltkreisen Raum- und Leistungsbedarf sinken weiter und werden grob geschätzt, wiederum um den Faktor 0,1 geringer	$\sim \frac{V_0}{100}$ $= V_0 \cdot 10^{-2}$ Gestellreihe	$\sim \frac{P_0}{100}$ $= P_0 \cdot 10^{-2}$ 2 kW
Ab 1968	Beginn der Großintegration Einchip-Taschenrechner			
Ab 1971/72	Mikroprozessoren Mikrorechner $\approx 10^4$ Transistoren auf einer Chipfläche von 4 mm × 6 mm	EDVA mit großintegrierten Schaltkreisen	$\sim V_0 \cdot 10^{-3}$ Einzelgestell	$\sim P_0 \cdot 10^{-3}$ 200 W
Ab 1977	Bauelemente in VLSI-Technik mehr als 10000 Transistoren auf einem Halbleiterchip	EDVA mit VLSI-Schaltkreisen	$\sim V_0 \cdot 10^{-4}$ Bücherkiste	$\sim P_0 \cdot 10^{-4}$ 20 W
Ab 1984	Bauelemente in ULSI-Technik mehr als 10^5 Transistoren auf einem Chip	EDVA mit ULSI-Schaltkreisen	$\sim V_0 \cdot 10^{-5}$ Schuhkarton	$\sim P_0 \cdot 10^{-5}$ 2 W

Seit etwa 1940 wurden auch Elektronenröhren als kontaktgebende Schaltelemente eingesetzt. Einer der ersten elektronischen Digitalrechner war der in den Jahren 1943 bis 1946 von *John Mauchly* und *Prosper Eckert* (beide USA) entwickelte ENIAC. Er enthielt etwa 18000 Elektronenröhren, benötigte einen kleinen Saal zur Aufstellung und eine Leistung von etwa 180 kW.
Mit der Erfindung des Transistors kam ein neuer Konkurrent für elektronische Schalter auf den Markt. Das Volumen und die benötigte Leistung vergleichbarer Rechner konnten gegenüber dem Röhrenrechner auf etwa je ein Zehntel gesenkt werden. **(Tafel 2.14)**.
Mit der Einführung *integrierter Schaltkreise* wurde eine weitere Verringerung des Volumens und der elektrischen Leistung erzielt. Nach **Tafel 2.15** unterscheidet man dabei verschiedene Integrationsgrade, wie SSI (small scale integration, Kleinintegration), MSI (medium scale integration, Mittelintegration), LSI (large scale integration, Großintegration), VLSI (very large scale integration) und ULSI (ultra large scale integration).

Tafel 2.15
Integrationsgrade elektronischer Schaltkreise

Integrationsgrad	Anzahl der integrierten Elemente auf einem Chip	Abkürzende Bezeichnung
1	1 ... 10	SSI
2	10 ... 100	SSI
3	10^2 ... 10^3	MSI
4	10^3 ... 10^4	LSI
5	10^4 ... 10^5	VLSI
6	$> 10^5$	ULSI

Man rechnet nicht allzu falsch, wenn man annimmt, daß je Integrationsgrad wieder eine Senkung von etwa je einem Zehntel an Volumen und Leistung gegenüber der vorhergehenden Stufe eingetreten ist.
Durch die Integration elektronischer Bauelemente konnte also eine vorher unvorstellbare Verringerung an benötigtem Volumen und benötigter elektrischer Leistung erzielt werden.
Wir wollen daher die so anschauliche Darstellung logischer Schaltungen mit Kontakten verlassen, Elektronenröhrenschaltungen nicht behandeln und von der Schalterwirkung der bipolaren und unipolaren Transistoren gleich zu den integrierten Schaltkreisen übergehen.

2.1.5. Der Transistor als Schalter

Transistoren sind steuerungsfähige Halbleiterbauelemente. Mit geringen Eingangsenergien können sie größere Ausgangsenergien steuern. Im *Schalterbetrieb* benötigt man nur zwei Zustände, die wahlweise eingenommen werden können. Beim Übergang von dem einen in den anderen Zustand kommt es durch die damit verbundenen Ladungsträgertransporte zu Verzögerungszeiten. Diese sind aber wesentlich geringer als beim Relais. Während Relais Schaltzeiten von einigen Millisekunden aufweisen, liegen die Verzögerungszeiten von Transistoren bei einigen Nanosekunden ($1\ \text{ns} = 10^{-9}\ \text{s}$). Transistoren schalten folglich um den Faktor 10^6 schneller als Relais.
Diese hohe *Schaltgeschwindigkeit* und die Möglichkeit, Transistoren mit der Längenausdehnung von etwa einem Mikrometer herzustellen, hat den Siegeszug der Halbleitertechnik in der Informationsverarbeitung bewirkt. Hinzu kommt noch, daß die Zuverlässigkeit von fehlerfrei gefertigten integrierten Schaltkreisen wesentlich größer ist als die von vergleichbaren Schaltungen in verdrahteter Ausführung, also mit Löt- oder sonstigen Verbindungen.
Allerdings bereiten die integrierten Schaltkreise einige Sorgen bei der Herstellung. Die benötigten Produktionsmittel sind teuer, und die Quote der fehlerfrei gefertigten Schaltkreise nimmt mit der Höhe des Integrationsgrads ab. Mit anderen Worten: Es ist außerordentlich schwierig, hochintegrierte Schaltkreise fehlerfrei zu fertigen.
Über die Wirkungsweise von Halbleiterbauelementen sind umfangreiche Bücher geschrieben wor-

den. Das liegt zum einen darin begründet, daß die atomaren Vorgänge im Innern der Halbleitermaterialien recht kompliziert sind. Zum anderen ist eine sehr große Palette technologischer Varianten entwickelt worden. Es ist daher an dieser Stelle nur ein kurzer Einblick möglich.
Es gibt zwei grundlegende Stromleitungsarten bei Halbleitern zu unterscheiden: *Eigenleitung* und *Störstellenleitung*.

- Als Eigenleitung bezeichnet man den Stromfluß durch Halbleitermaterial (z. B. Si, Ge) mit höchstem Reinheitsgrad. In einem solchen Material können nur merkliche Ströme fließen, wenn durch Wärmeenergiezufuhr (höhere Temperaturen) bewegliche Ladungsträger erzeugt werden.
- Von Störstellenleitung spricht man bei Halbleitermaterial mit höchstem Reinheitsgrad, dem eine entsprechende Dosis von Fremdatomen zugesetzt ist. Man wählt solche Fremdatome, die entweder ein Bindungselektron mehr oder weniger als die Atome des Halbleitermaterials aufweisen. Dadurch treten Ladungsträger auf, die nicht zur Bindung zwischen den einzelnen Atomen benötigt werden und daher schon bei geringer Energiezufuhr beweglich sind. Bei der Störstellenleitung werden wieder zwei Fälle unterschieden:
  - Haben die Fremdatome ein Valenzelektron je Atom mehr als das Halbleitermaterial, so werden diese Elektronen nicht zu Bindungszwecken benötigt und sind schon bei normaler Temperatur leicht beweglich. Man spricht dann von *n*-leitendem Halbleitermaterial.
  - Haben die Fremdatome ein Valenzelektron je Atom weniger, so sind sie bereit, die zur Bindung fehlenden Elektronen von anderen Gitterplätzen abzuziehen, wobei dann dort jeweils ein positives Loch entsteht. Diese positiven Löcher sind ortsveränderlich, also ebenfalls beweglich. Man spricht dann von *p*-leitendem Halbleitermaterial.

Durch sinnvolle Kombinationen von p-leitenden und n-leitenden Halbleiterschichten, Isolierschichten sowie metallischen Elektroden und Verbindungsleitungen können die verschiedensten Effekte erzielt werden. Technologisch wird dieser Prozeß durch gezieltes Bedampfen mit speziellen Gasen realisiert, wobei die jeweils nicht zu bedampfenden Flächen durch Masken abgedeckt werden müssen. Aus dem Gas scheiden sich die benötigten Atome aus und erzeugen im Halbleitersubstrat die erforderlichen n- oder p-leitenden Schichten. Außerdem spielen noch Ätz- und Abtrageprozesse eine Rolle.
Bei den so entstehenden Transistoren unterscheidet man *bipolare* und unipolare Typen. Beim Unipolartransistor ist die Richtung des Ausgangsstroms beliebig, d. h., es können die beiden Ausgangselektroden vertauscht werden.

2.1.5.1. Bipolartransistor

Bild 2.12a zeigt den schematischen Aufbau eines npn-Bipolartransistors. Bild 2.12b zeigt das Schaltsymbol mit äußerer Beschaltung zur Steuerung eines Stromes. Im **Bild 2.13** ist das Kennlinienfeld dargestellt, das die funktionale Abhängigkeit der einzelnen Stromstärken und Spannungen des Transistors demonstriert.

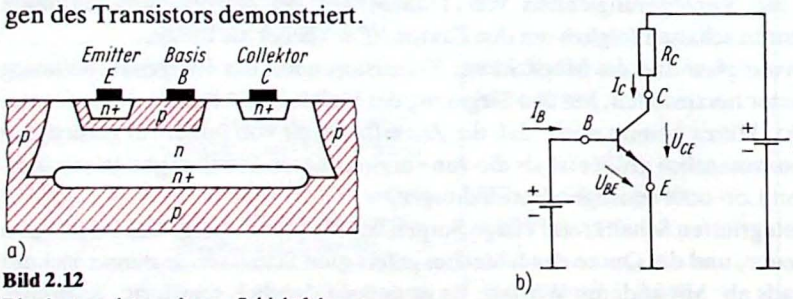


Bild 2.12
Bipolartransistor mit npn-Schichtfolge
a) schematischer Aufbau; b) Gundschaltung mit Kollektorwiderstand R_C und Gleichspannungsversorgung zur Arbeitspunkteinstellung

Vergrößert man die Spannung U_{BE} von 0 V an (Arbeitspunkt A_1), so nimmt auch die Stromstärke I_C, von $I_C = 0$ beginnend, immer größere Werte an. Ab der Spannung U_{BEsat} nimmt die Stromstärke I_C nicht mehr zu, da dann alle bei der geringen Spannung U_{CEsat} zum Kollektor treibenden Ladungsträger bereits am Stromfluß beteiligt sind (Sättigungszustand). Der Arbeitspunkt liegt dann bei A_2.

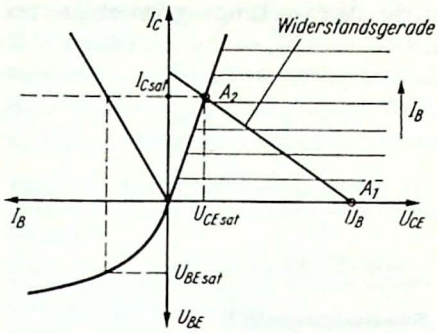


Bild 2.13. Vereinfachtes Kennlinienfeld eines npn-Bipolartransistors

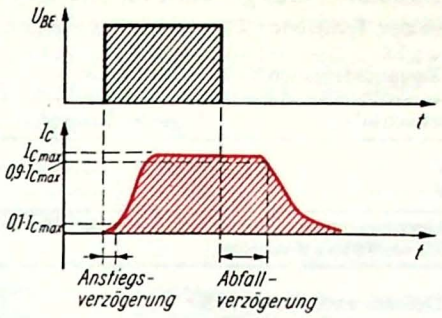


Bild 2.14. Verzögerungszeiten beim Schalterbetrieb eines Transistors

Im Schalterbetrieb wird der Transistor wechselweise vom Arbeitspunkt A_1 (Aus-Zustand) zum Arbeitspunkt A_2 (Ein-Zustand) und zurück gesteuert. Wird dabei U_{BEsat} zu stark überschritten, werden unnötig viele Ladungsträger in der Basiszone gespeichert, deren Abbau beim Zurückschalten eine meist nicht gewünschte Verzögerungszeit ergibt **(Bild 2.14)**.

Als Beispiel einer in Bipolartechnik hergestellten logischen Schaltung sei ein NAND-Gatter vorgestellt **(Bild 2.15)**. Liegen beide Eingänge E1 und E2 auf hohem Potential, so ist die Spannung U_{BE} des Transistors T1 niedrig, und er läßt keinen Emitterstrom fließen. Von der Basis des Transistors T1 kann jedoch ein Strom I_{C1} über den Kollektor von T1 zur Basis von T2 fließen. Dadurch wird T2 aufgesteuert, und durch den Spannungsabfall an R3 wird auch T4 in den eingeschalteten Zustand gebracht. Damit ist das Potential am Ausgang A niedrig. In diesem Zustand ist T3 wegen des Spannungsabfalls an R2 (niedriges Basispotential an T3) gesperrt.

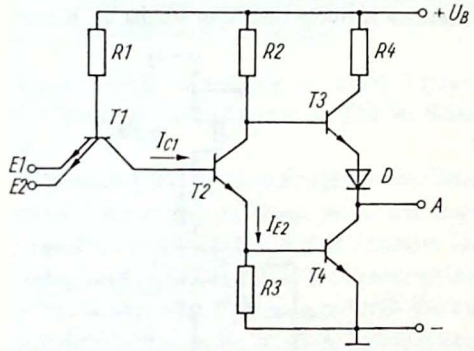


Bild 2.15
NAND-Gatter in Bipolartechnik

Liegt ein oder liegen beide Emitter E1 und E2 auf geringem Potential, so wird T1 aufgesteuert. Damit fließt I_{C1} in der umgekehrten Richtung als im Bild 2.15 gezeichnet. Das bedeutet aber Sperrung von T2 und wegen $I_{E2} = 0$ auch Sperrung von T4. Da nun über R2 kein wesentlicher Spannungsabfall auftritt, wird T3 (hohes Basispotential) geöffnet und damit der Ausgang A auf hohes Potential gebracht.

In der TTL-Technik (TTL Transistor-Transistor-Logik), zu der das vorgestellte NAND-Gatter gehört, hat man die logischen Zustände gemäß **Tafel 2.16** definiert (positive Logik). Wir wollen noch die Definitionen nach **Tafel 2.17** einführen. Mit den Tafeln 2.16 und 2.17 bekommt die Schaltung

nach Bild 2.15 die in **Tafel 2.18** dargestellte Schaltbelegungstabelle. Wie man erkennt, entsteht die NAND-Funktion $y = \overline{x_1 x_2}$.
Als praktische Merkhilfe für die Wirkungsweise des NAND-Gatters kann man folgenden Satz verwenden: Der Zustand 0 einer Eingangsvariablen erzwingt den Ausgangszustand 1 (0 greift durch). Das soll heißen: Sowie der Zustand 0 nur bei einer Eingangsvariablen auftritt, erscheint am Ausgang der Zustand 1. Das gilt auch für NAND-Gatter mit mehr als zwei Eingangsvariablen, bei denen dann der Transistor T1 noch weitere Emitter aufweist.

Tafel 2.16. Zustandsdefinition für TTL-Schaltkreise

Spannungsbereich in V	Logischer Zustand
0 ... 0,4 V	0
2,4 ... 5,5 V	1

(Gilt für Schaltkreisausgänge; für Schaltkreiseingänge ist der verbotene Zwischenbereich von 0,8 bis 2 V eingeengt.)

Tafel 2.17. Definition von Variablen für das NAND-Gatter nach Bild 2.15

Klemme	Zustandsvariable
E1	x_1
E2	x_2
A	y

Tafel 2.18. Schaltbelegungstabelle für das NAND-Gatter

x_2	x_1	y
0	0	1
0	1	1
1	0	1
1	1	0

2.1.5.2 Unipolartransistor

Beim Unipolartransistor ist die Steuerelektrode (Gate) durch eine Isolierschicht vom Halbleitermaterial getrennt. Infolge dieser galvanischen Trennung vollzieht sich die Steuerung nunmehr über die Wirkung eines elektrischen Feldes im Nichtleiter. Man spricht daher auch von *Feldeffekttransistoren*. Bei tiefen Schaltfrequenzen ist die Steuerstromstärke (Verschiebungsstromstärke) sehr gering, so daß dann hohe Eingangswiderstände wirksam werden.
Bild 2.16 zeigt die Prinzipdarstellung eines MOS-Transistors vom p-Kanal-Anreicherungstyp, sein verallgemeinertes Schaltsymbol, ein Schaltbild zur Schalteranwendung und das statische Kenn-

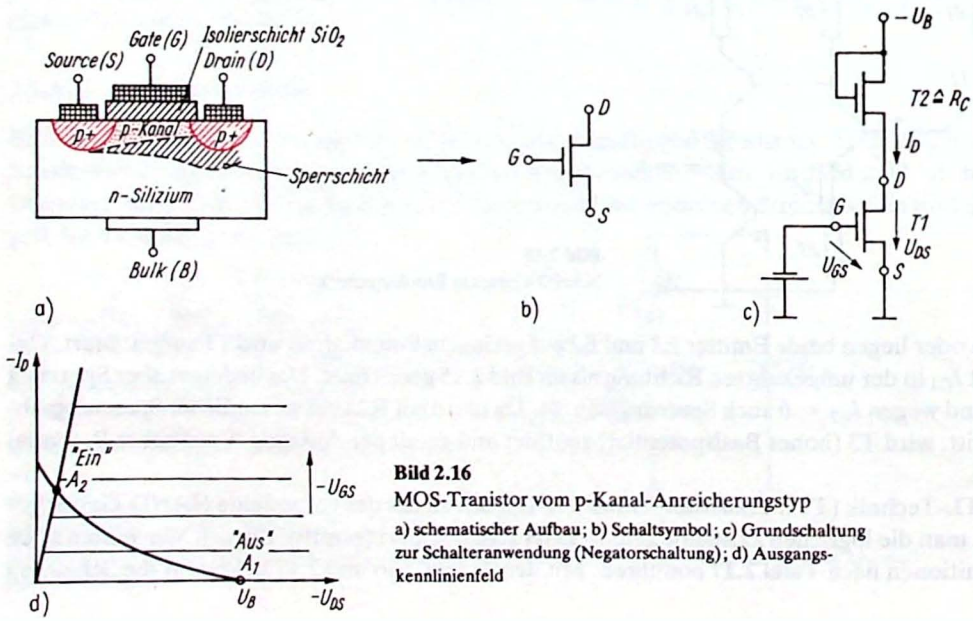


Bild 2.16
MOS-Tranistor vom p-Kanal-Anreicherungstyp
a) schematischer Aufbau; b) Schaltsymbol; c) Grundschaltung zur Schalteranwendung (Negatorschaltung); d) Ausgangskennlinienfeld

linienfeld. Eine Besonderheit stellt im Bild 2.16c der Transistor T2 dar. Da sein Gate mit einer der beiden anderen Elektroden verbunden ist, wirkt dieser Transistor wie eine im Durchlaßbereich betriebene Diode. T2 wirkt also nur wie ein Widerstand, allerdings mit nichtlinearer Kennlinie. Daher ist die Strom-Spannungs-Kennlinie des aktiven Zweipols (sonst die Widerstandsgerade) hier leicht gekrümmt. Diese Darstellung von Widerständen durch Transistordioden wird wegen der besser beherrschbaren Technologie vorgenommen.

Der Schaltzustand von MOS-Schaltkreisen des p-Kanal-Anreicherungstyps wird in *negativer Logik* beschrieben. **Tafel 2.19** gibt zu den definierten Pegelverhältnissen einen Überblick. Mit abnehmender negativer Spannung U_{GS} fällt die Stromstärke $-I_D$ bei dem im Bild 2.16 vorgestellten Transistor, so daß er immer weiter in Sperrichtung getrieben wird.

Tafel 2.19. Zustandsdefinitionen für einen MOS-Negator vom p-Kanal-Anreicherungstyp

Eingang		Arbeitspunkt	Ausgang	
U_{GS}	logischer Zustand		U_{DS}	logischer Zustand
Schwach negativ	0	A1 (Aus)	stark negativ	1
Stark negativ	1	A2 (Ein)	schwach negativ	0

Tafel 2.20. Schaltbelegungstabellen für die Schaltungen nach Bild 2.17

x_2	x_1	y
0	0	1
0	1	1
1	0	1
1	1	0

a) für die NAND-Schaltung nach Bild 2.17 a

x_2	x_1	y
0	0	1
0	1	0
1	0	0
1	1	0

b) für die NOR-Schaltung nach Bild 2.17 b

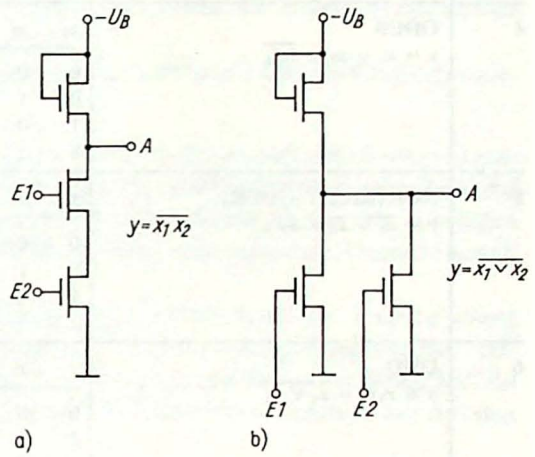


Bild 2.17
Elementare Logikschaltungen mit MOS-Transistoren
a) NAND-Glied; b) NOR-Glied

Eine NAND-Schaltung mit MOS-Transistoren zeigt **Bild 2.17a**. Eine NOR-Schaltung ist im **Bild 2.17b** dargestellt. **Tafel 2.20** gibt die Schaltbelegungstabelle unter Berücksichtigung der Tafel 2.19 an.

Wenn einer der beiden Eingänge der Schaltung nach Bild 2.17a niedriges negatives Potential (logischer Zustand 0) aufweist, so ist der zugehörige Transistor gesperrt, und am Ausgang liegt hohes negatives Potential (logischer Zustand 1). Erst wenn stark negatives Potential an E1 und E2 gleichzeitig liegt, werden beide Transistoren leitend, so daß das Potential des Punktes A schwach negativ wird. Damit liegt hier eine NAND-Verknüpfung vor.

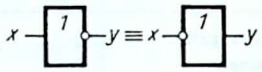
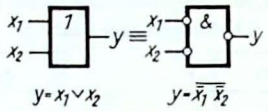
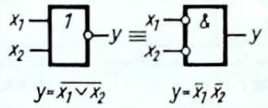
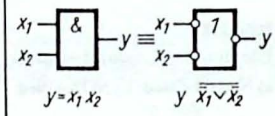
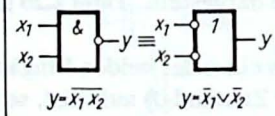
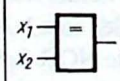
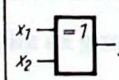
Ähnlich läßt sich die NOR-Schaltung nach Bild 2.17b erläutern. Hier befindet sich der Ausgang A nur dann im Zustand 1, wenn beide Transistoren gesperrt sind, d. h. an E1 und E2 gleichzeitig der Zustand 0 auftritt.

2.1.6. Übergang zu allgemeinen Logikschaltungssymbolen

Im Abschnitt 2.1.5. ist vielleicht trotz der knappen und elementaren Beschreibung zu erkennen gewesen, daß bei der praktischen Realisierung elektronischer Logikschaltungen etliche Probleme zu lösen sind.

Neben den Potential- und Zustandsverhältnissen sind vom Entwickler die Verzögerungszeiten der einzelnen Bauelemente, das ordnungsgemäße Zusammenwirken aller Baugruppen, die gegenseitige Störbeeinflußung und der Einfluß äußerer Störungen zu beachten. Oftmals sind komplizierte Situationen zu beherrschen.

Tafel 2.21. Auszugsweise Darstellung von Schaltzeichen logischer binärer Elemente nach TGL 16056/03

Nr.	Bezeichnung und Funktion	Schaltbelegungstabelle	Schaltungssymbol
1	Einsfunktion $y = 1$ Einsgenerator	y 1	
2	Nullfunktion $y = 0$ Nullgenerator	y 0	
3	Negator $y = \bar{x}$	x | y 0 | 1 1 | 0	
4	ODER $y = x_1 \vee x_2 = \overline{\bar{x}_1 \bar{x}_2}$	x_1 x_2 | y 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	 $y = x_1 \vee x_2$ $y = \overline{\bar{x}_1 \bar{x}_2}$
5	NOR (NICHT–ODER) $y = \bar{x}_1 \vee \bar{x}_2 = \bar{x}_1 \bar{x}_2$	x_1 x_2 | y 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	 $y = \overline{x_1 \vee x_2}$ $y = \bar{x}_1 \bar{x}_2$
6	UND $y = x_1 x_2 = \overline{\bar{x}_1 \vee \bar{x}_2}$	x_1 x_2 | y 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	 $y = x_1 x_2$ $y = \overline{\bar{x}_1 \vee \bar{x}_2}$
7	NAND (NICHT–UND) $y = \overline{x_1 x_2} = \bar{x}_1 \vee \bar{x}_2$	x_1 x_2 | y 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	 $y = \overline{x_1 x_2}$ $y = \bar{x}_1 \vee \bar{x}_2$
8	Äquivalenz $y = \bar{x}_1 \bar{x}_2 \vee x_1 x_2$	x_1 x_2 | y 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1	
9	Antivalenz EXCLUSIV-ODER $y = x_1 \bar{x}_2 \vee \bar{x}_1 x_2$	x_1 x_2 | y 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	

Um wieder einen gangbaren Weg zur Darstellung des Stoffes im vorgesehenen Rahmen zu finden, sollen diese Probleme in stark abgerüsteter Form betrachtet werden. Wir wollen daher Schaltungseinzelheiten nur dort behandeln, wo sie unbedingt für das Verständnis benötigt werden. In allen anderen Fällen sollen *Signalflußbilder* (Blockbilddarstellungen) eingesetzt werden.
Die Grundlage für die Darstellung von Schaltzeichen logischer binärer Elemente bildet TGL 16056. In **Tafel 2.21** sind die wichtigsten logischen Grundfunktionen und ihre standardgerechten Schaltungssymbole dargestellt. Bis zu Nr. 7 sind uns ihre Kontaktdarstellungen bekannt. Die standardgerechten Schaltungssymbole bestehen aus einem rechteckigen Kästchen, in dem ein entsprechendes Symbol (1, &, =, = 1) die Art der Grundschaltung kennzeichnet.
Wird eine Eingangsvariable negiert verwendet, so wird dies durch einen kleinen Kreis am entsprechenden Eingang auf dem Kästchenrand gekennzeichnet. Soll eine Ausgangsvariable negiert werden, muß der kleine Kreis am entsprechenden Ausgang dargestellt werden.
Die Kästchen sollen in allgemeiner Form als *Glieder* bezeichnet werden. Demzufolge sprechen wir auch von UND-Glied, NOR-Glied usw.
Noch nicht behandelt haben wir die logischen Schaltungen nach Nr. 8 und 9. Dazu folgende allgemeine Aussagen:

- Eine Äquivalenzschaltung belegt die Ausgangsvariable y mit 1, wenn alle Eingangsvariablen jeweils gleich sind, d. h. entweder alle 0 oder alle 1 sind.
  Eine Antivalenzschaltung belegt die Ausgangsvariable y mit 1, wenn nur eine der Eingangsvariablen mit 1 belegt ist.

Mit den in Tafel 2.21 dargestellten allgemeinen Schaltungssymbolen kann man vollständige Logikschaltungen in Form eines Signalflußbilds darstellen. Diese Signalflußbilder beschreiben das logische Verhalten der Schaltung mathematisch exakt und sind unabhängig von der jeweiligen materiellen Realisierungsvariante, wie z. B. durch Relais, bipolare oder unipolare Transistorschaltungen.
Das hat den Vorteil, daß man sich bei dieser Darstellungsform nicht um Probleme der Anpassung bezüglich Spannungs- oder Stromstärkewerten oder der elektrischen Leistungsaufnahme oder der Eingangs- und Ausgangswiderstände kümmern muß. Wir brauchen uns bei einem Signalflußbild also nicht mit Pegelwerten wie hoch (high) oder tief (low) zu beschäftigen, sondern nur mit den logischen Zuständen 0 oder 1.
Um in den Fällen, in denen es doch notwendig ist, mit Potential- (Pegel-) Werten zu arbeiten, die Anschlußbedingungen klären zu können, wollen wir für unsere Signalflußbilddarstellung folgendes vereinbaren:

- Der logische Zustand 1 soll einem positiven Potential (ungefähr gleich der Batteriespannung $+U_B$) entsprechen.
  Der logische Zustand 0 soll mit dem Potential 0 V identisch sein.

Das heißt z. B., daß wir an alle Eingänge eines UND-Gliedes die Spannung $+U_B$ legen müssen, wenn der Ausgang ebenfalls $+U_B$ (also $y = 1$) aufweisen soll.

2.1.7. Wichtige Grundschaltungen

Mit den vorgestellten logischen Grundfunktionsgliedern lassen sich kombinatorische Schaltungen aufbauen. Im folgenden wollen wir solche betrachten, die wichtige Funktionen bei der Informationsverarbeitung zu realisieren gestatten.

2.1.7.1. Auswahl eines Logikzustands auf einer Leitung

Zur Auswahl des Zustands 0 oder 1 auf einer Leitung benötigt man im Prinzip einen Schalter, der entweder offen (Zustand 0) oder geschlossen (Zustand 1) ist und damit die notwendige Spannung an die Leitung schaltet (Bild 2.18).

Bei elektronisch gesteuerten Systemen muß an die Stelle des Schalterkontakts ein Transistor oder ein standardisiertes Gatter treten.
Wir wollen hier nicht die Problematik behandeln, die dann eine Rolle spielt, wenn auf der Leitung infolge hoher Schaltfrequenz oder steiler Impulsflanken Ausgleichsvorgänge (Überlagerung von hin- und rücklaufender Welle) auftreten. Dazu würde es ausführlicher Erörterungen bedürfen. Wir wollen annehmen, daß die Leitung so kurz ist, daß der Widerstandsanpassung und der Schaltfrequenz keine Beachtung geschenkt werden müssen.

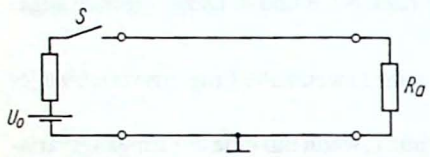


Bild 2.18. Auswahl des Logikzustands einer Leitung durch Schalterkontakt

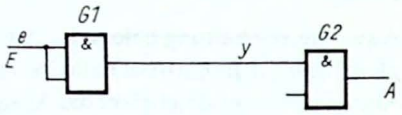


Bild 2.19. Auswahl des Logikzustands einer Leitung durch ein UND-Glied

Bild 2.19 zeigt die Verbindung zweier UND-Glieder über eine Leitung (einpolige Darstellung, Masseleitung nicht mitgezeichnet). Infolge der Zusammenschaltung der zum Gatter G1 gehörenden Eingänge erhält die Verbindungsleitung den Zustand $y = 1$, wenn am Eingang E ebenfalls $e = 1$ ist, bzw. $y = 0$, wenn $e = 0$ ist.
Es kann nun auch der Fall eintreten, daß mehrere Leitungen vorliegen, von denen jeweils eine wahlweise in den Zustand $y = 1$ gebracht werden soll. Als Beispiel wollen wir ein System mit vier Leitungen wählen. Dieses Problem kann man wie im **Bild 2.20** lösen, indem man für jede der Leitungen einen Schalter vorsieht und verabredet, daß immer nur ein Schalter geschlossen sein darf.
Man kann das Problem aber auch durch ein Adressierungssystem (vgl. Bild 2.11) lösen. Dazu benötigt man eine logische Schaltung, die auf eine bestimmte Bitkombination die richtige Leitung in den Zustand 1 bringt. Für das Beispielsystem mit den vier Leitungen wollen wir als Adressen die Binärzahlen von null bis drei verwenden. Dazu soll die Schaltbelegungstabelle nach **Tafel 2.22** definiert werden.

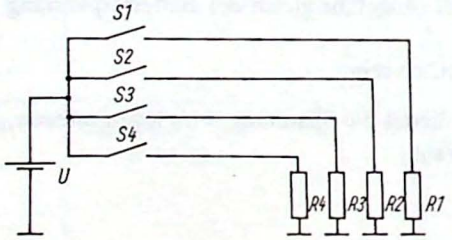


Bild 2.20
Wahlweises Einschalten des Logikzustands 1 auf einer von vier Leitungen durch Schalterkontakt

Tafel 2.22
Schaltbelegungstabelle für das Signalflußbild nach Bild 2.21

Adresse		Leitungszustand			
e_2	e_1	y_1	y_2	y_3	y_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

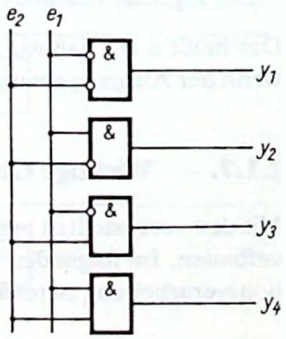


Bild 2.21
Wahlweises Einschalten des Logikzustandes 1 auf einer von vier Leitungen durch adressierbare Gatter

Die Adresse wird aus der Binärzahl $z = e_2 \cdot 2^1 + e_1 \cdot 2^0$ gebildet. Bei $z = 10$ B gilt z. B., daß $y_3 = 1$ und alle anderen y-Werte gleich Null sind, daß folglich die Leitung drei in den Zustand 1 geschaltet wird. *Überzeugen Sie sich bitte davon, daß Tafel 2.22 so definiert ist, daß bei Erhöhung der Adresse um eins gerade die Leitung mit dem nächsthöheren Index eingeschaltet wird.* **Bild 2.21** stellt das zugehörige Signalflußbild dar.

2.1.7.2. Signaleinschalter (Torschaltungen)

Eine andere wichtige Grundschaltungsfunktion besteht darin, ein Binärsignal nach Bedarf auf eine Leitung zu schalten. Solche Schaltungen bezeichnet man auch als Torschaltungen.

Bild 2.22 zeigt zunächst die Problemlösung mit einem Kontaktschalter. Diesmal soll also keine gleichbleibende Spannung durch den Schalter angelegt werden, sondern bei geschlossenem Kontakt kann sich das Eingangssignal beliebig ändern. Das Eingangssignal stellt eine zeitlich frei veränderliche Zustandsvariable x dar.

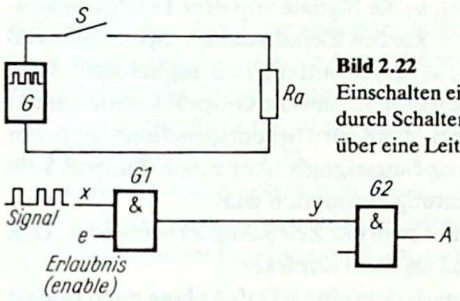


Bild 2.22
Einschalten eines Binärsignals durch Schalterkontakt zur Übertragung über eine Leitung

Bild 2.23. Einschalten eines Binärsignals durch ein UND-Glied (Tor) zur Übertragung über eine Leitung

Tafel 2.23. Schaltbelegungstabelle für das Signalflußbild nach Bild 2.23

e	x	y	
0	0	0	$y = 0$
0	1	0	
1	0	0	$y = x$
1	1	1	

Die elektronische Lösung für eine Leitung sieht ähnlich wie die im Bild 2.19 aus, nur daß jetzt die beiden Eingänge des Gatters G1 getrennt benutzt werden **(Bild 2.23, Tafel 2.23)**. Mit dem Signal $e = 1$ wird sozusagen dem Signal x die Erlaubnis erteilt, auf die Leitung zu gelangen. Ist $e = 1$, so entspricht $y = x$. Bei $e = 0$ ist unabhängig vom Zustand der Variablen x die Variable $y = 0$.

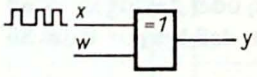
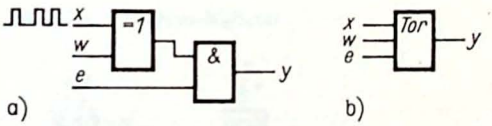


Bild 2.24. EXCLUSIV-ODER-Glied zur wahlweisen Übertragung des Signals in identischer oder negierter Form

Tafel 2.24. Schaltbelegungstabelle für Bild 2.24

w	x	y	
0	0	0	$y = x$
0	1	1	
1	0	1	$y = \bar{x}$
1	1	0	



a) Signalflußbild
b) Schaltsymbol

Bild 2.25. Glied zum Einschalten eines Signals in identischer oder negierter Form

Mitunter liegt die Forderung vor, anstelle des Signals x das negierte Signal $\bar{x}$ auf die Leitung zu schalten. Zum wahlweisen Einschalten von x oder $\bar{x}$ muß man eine Schaltung entwerfen, die die Schaltbelegungstabelle nach **Tafel 2.24** realisiert. Nach dieser Kombinationstabelle wird für $w = 0$ die direkte Verknüpfung $y = x$ und für $w = 1$ die negierte Verknüpfung $y = \bar{x}$ wirksam. Vergleicht man Tafel 2.24 mit Tafel 2.21, so erkennt man, daß w, x, y nach einer Antivalenzschaltung oder EXCLUSIV-ODER (EXOR) verknüpft sein müssen. **Bild 2.24** zeigt das Blockbild der die Tafel 2.24 realisierenden Schaltung.
Mit einem zusätzlichen Erlaubniseingang (enable) erhält man **Bild 2.25.**

2.1.7.3. Signalumschalter (Multiplexer, Demultiplexer)

Oftmals besteht die Aufgabe, Signale aus mehreren Leitungen oder Gliedausgängen nacheinander abzutasten und auf eine Leitung zu schalten.

Multiplexer. Eine solche Aufgabenstellung erhält man z. B. bei der zeitgeschachtelten Übertragung von mehreren Signalen zur Einsparung von Leitungen **(Bild 2.26)**.

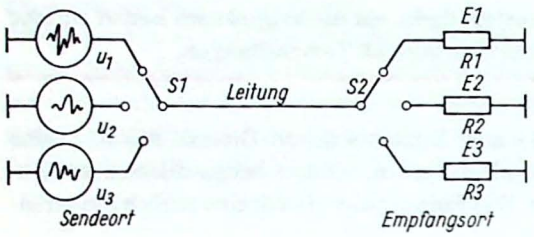


Bild 2.26
Vereinfachte Darstellung einer Signalumschaltereinrichtung zur zeitgeschachtelten Übertragung analoger Signale

Nehmen wir an, daß im Bild 2.26 die drei Spannungen u_1, u_2, u_3 die Signale von drei Telefongesprächen darstellen und daß der rotierende Schalter S1 in sehr kurzen Zeitabständen Sprachschwingungsmomentanwerte in der Reihenfolge $u_1, u_2, u_3, u_1, u_2, u_3, u_1, \ldots$ auf die Leitung schaltet. Gibt man am Empfangsort einen synchron zu S1 umlaufenden Schalter S2, der die Gesprächsteile wieder in richtiger Reihenfolge auf die Empfänger E1 bis E3 schaltet, dann wird bei entsprechend schneller Rotation der Schalter S1 und S2 und bei Filterung des Empfangssignals über einen Tiefpaß kein Empfänger merken, daß er nicht die ganze Zeit mit der Leitung verbunden war.
Einen solchen leitungssparenden Übertragungsbetrieb nennt man ein *Zeitmultiplexverfahren*. Den Schalter S1 bezeichnet man als *Multiplexer*, den Schalter S2 als *Demultiplexer*.
In der EDV nennt man das ähnlich arbeitende Verfahren, nach dem eine EDV-Anlage nach Bedarf zeitgeschachtelt mehreren Kunden gleichzeitig ihre Rechenprogramme abzuarbeiten gestattet, *Time-sharing*. Der Unterschied zum Zeitmultiplexverfahren der Nachrichtenübertragung besteht vor allem darin, daß bei dieser Leitungen, in der EDV aber Rechner eingespart werden. Auf alle Fälle sorgen beide Verfahren für eine Mehrfachausnutzung.
Zum Kennenlernen der Anwendung des Zeitmultiplexverfahrens auf binäre Signale wollen wir als erstes Beispiel einen 2 : 1-Multiplexer entwerfen, den man auch als *Signalumschalter* bezeichnen kann. In Abhängigkeit von einer Steuervariablen e soll entweder das Signal x_1 oder das Signal x_2 auf die Leitung gelangen. In der Schaltbelegungstabelle **Tafel 2.25** legen wir fest, daß bei $e = 0$ das Signal x_1 und bei $e = 1$ das Signal x_2 übertragen wird.

e	y
0	x_1
1	x_2

Tafel 2.25
Schaltbelegungstabelle für das Bild 2.27

Die Tafel 2.25 läßt sich in folgende Gleichung der Schaltalgebra überführen:

$$y = \bar{e}x_1 \vee ex_2.$$

Die Realisierung als Blockbild zeigt **Bild 2.27**.

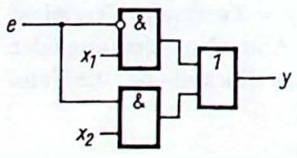


Bild 2.27
2:1-Multiplexer für Binärsignale

Ein 4 : 1-Multiplexer **(Bild 2.28)** muß zwei Adreßvariable e_1, e_2 zur Generierung von vier Auswahladressen aufweisen. Die Schaltbelegungstabelle entspricht der nach **Tafel 2.26**. Das Schaltsymbol für einen 4 : 1-Multiplexer zeigt **Bild 2.29**.

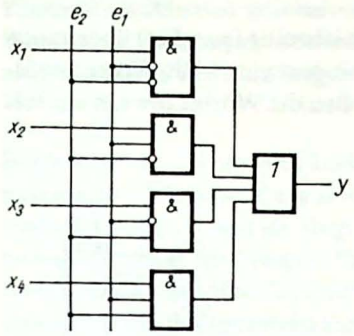


Bild 2.28. 4 : 1-Multiplexer

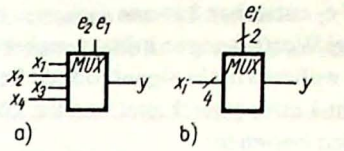


Bild 2.29
Schaltsymbole für 4 : 1-Multiplexer
a) bei Darstellung jeder Leitung; b) bei Darstellung von Leitungsbündeln

e_2	e_1	y
0	0	x_1
0	1	x_2
1	0	x_3
1	1	x_4

Tafel 2.26
Schaltbelegungstabelle für Bild 2.28

Demultiplexer. Ein Demultiplexer schaltet ein ankommendes Signal x auf eine ausgewählte Leitung. Man kann ihn auch als *Auswahlschaltung* für eine von mehreren Leitungen bezeichnen, wobei dann die ausgewählte Leitung das Signal überträgt.
Ein 4 : 1-Demultiplexer ist im **Bild 2.30** dargestellt. Seine Schaltbelegungstabelle **(Tafel 2.27)** ist ähnlich der von Tafel 2.22.
Nur wenn die entsprechende Adresse, gebildet aus e_1 und e_2, die Erlaubnis erteilt, kann das Signal x auf eine der Leitungen y_1 bis y_4 gelangen. Die anderen Leitungen behalten den Zustand 0.
Das Schaltsymbol für einen 4 : 1-Demultiplexer (DMUX) zeigt **Bild 2.31**.

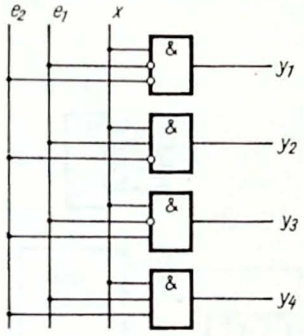


Bild 2.30. 1 : 4-Demultiplexer

e_2	e_1	y_1	y_2	y_3	y_4
0	0	x	0	0	0
0	1	0	x	0	0
1	0	0	0	x	0
1	1	0	0	0	x

Tafel 2.27
Schaltbelegungstabelle für Bild 2.30

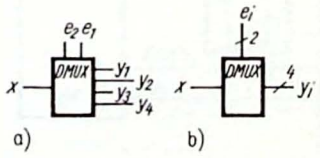


Bild 2.31. Schaltsymbole für 1 : 4-Demultiplexer
a) bei Darstellung jeder Leitung
b) bei Darstellung von Leitungsbündeln

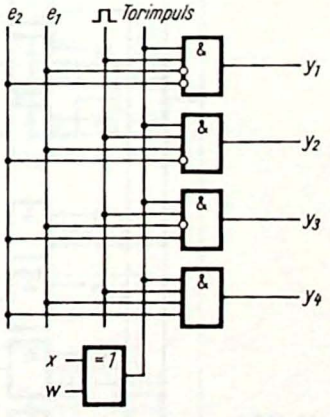


Bild 2.32
1 : 4-Demultiplexer mit Negationsmöglichkeit der Variablen x und Auswahlmöglichkeit des Übertragungsbeginns durch Torimpuls

Mit **Bild 2.32** wird noch der Signalflußplan für einen 1 : 4-Demultiplexer mit Negationsmöglichkeit der Variablen x vorgestellt. Außerdem ist diese Baugruppe mit einem zusätzlichen Eingang zur Anschaltung eines Torimpulses versehen. Damit können alle Tore gleichzeitig gesperrt werden bzw.

kann das adressierte Tor erst übertragen, wenn dieser Torimpuls vorliegt. Der Torimpuls wird dann gegeben, wenn das aus e_1 und e_2 bestehende Adreßsignal vollständig vorliegt. Dadurch werden die kurzzeitig falschen Adressen nicht wirksam, die durch die nie vollständige Gleichzeitigkeit beim Eintreffen von e_1 und e_2 entstehen können.

Wortumschaltung. Bei Wortleitungen müssen mehrere Bitstellen gleichzeitig (parallel) übertragen werden. Als Beispiel wollen wir ein Signalflußbild kennenlernen, mit dem vier 4-Bit-Wörter wahlweise auf eine 4-Draht-Leitung geschaltet werden können. Dazu sollen die Wörter jeweils aus folgenden Einzelvariablen bestehen:

Wort 1 $w_1 = x_{11}, x_{12}, x_{13}, x_{14}$

Wort 2 $w_2 = x_{21}, x_{22}, x_{23}, x_{24}$

usw. und allgemein:

Wort i $w_i = x_{i1}, x_{i2}, x_{i3}, x_{i4}$.

Dazu können wir auf den 4 : 1-Multiplexer zurückgreifen und müssen ihn nur, wie im **Bild 2.33** dargestellt, für jede Bitstelle wiederholt aufbauen.

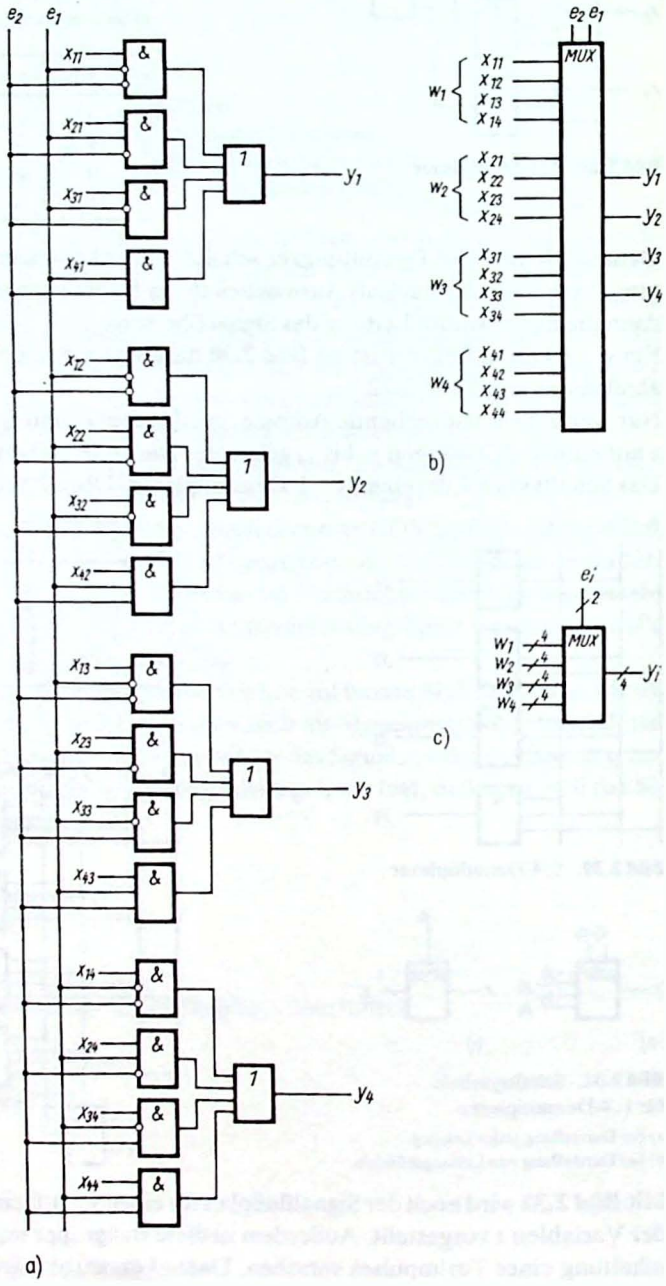


Bild 2.33
4 : 1-Multiplexer für vier 4-Bit-Wörter
a) Signalflußbild
b) Schaltsymbol mit einzelnen Leitungen
c) Schaltsymbol mit Leitungsbündeln

2.1.7.4. Sammelschienenübertragung (Bus)

Wir wissen: Das Busprinzip hat den Vorteil, daß die an dieser Sammelleitung angeschlossenen Bausteine wahlweise miteinander in Informationstausch treten können.
Durch eine Steuerzentrale wird jeweils einem Baustein die Sendeerlaubnis und einem oder mehreren Bausteinen die Empfangserlaubnis erteilt.

Betrachten wir hierzu **Bild 2.34**. Die Register 1 bis N sollen über den Datenbus ihre Informationen austauschen können. Dazu sind sie empfangsseitig direkt und sendeseitig über Tore mit dem Datenbus gekoppelt. Soll ein Register eine Kopie des Signalzustands auf dem Datenbus übernehmen, so muß es einen EE-Impuls (Erlaubnis zum Empfang) erhalten. Soll ein Register seinen Informationszustand über den Datenbus zu einem anderen Register übertragen, so muß sein Tor durch einen ES-Impuls (Erlaubnis zum Senden) geöffnet werden. Das Register N nimmt eine Sonderstellung ein. Es dient als Zwischenspeicher und braucht nicht adressiert zu werden.
Als Beispiel wollen wir uns ansehen, wie Daten vom Register 1 (G5) zum Register 2 (G9) übertragen werden. Nach Empfang des Befehls „Übertrage Daten vom Register 1 zum Register 2" legt die Steuerzentrale zunächst die Adresse des Senderegisters auf den Adreßbus. Dadurch erzeugt der Decoder G2 das Signal $e_1 = 1$. Werden nun noch $s_1 = 1$ und Takt 1 = 1, so bildet G3 das Signal ES = 1. Dadurch wird Tor 1 (G6) geöffnet, und der Datenbus erhält das Bitmuster vom Register 1.

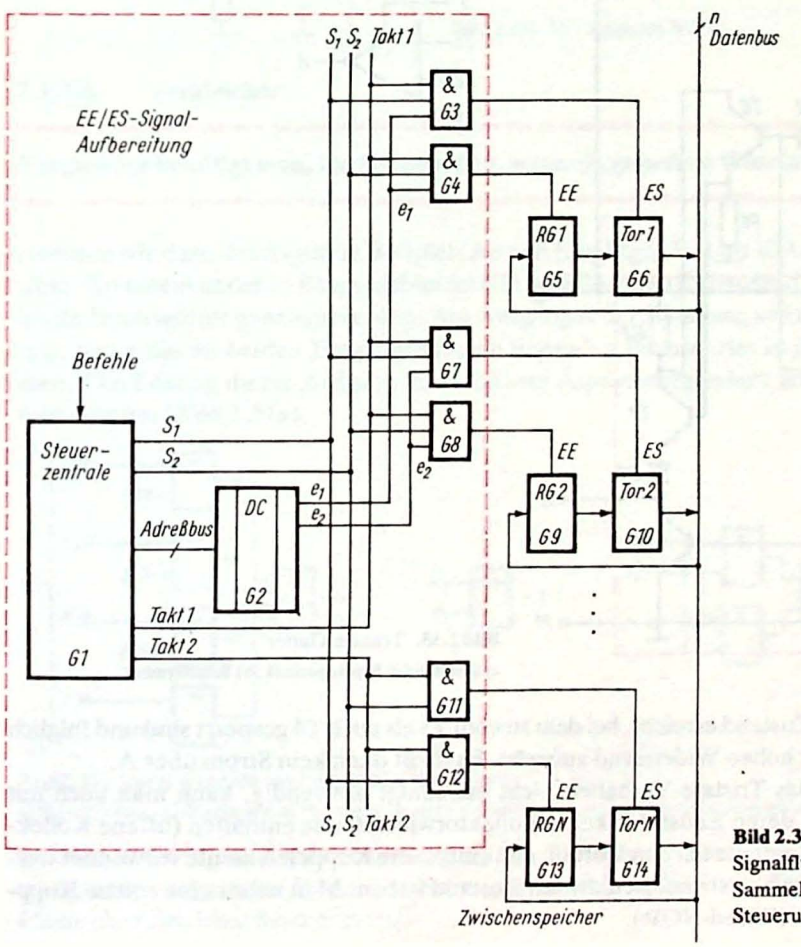


Bild 2.34
Signalflußbild für Sammelschienenübertragungs-Steuerung

Das Register N erhält nun über $s_1 = 1$ und Takt 2 = 1 das Signal EE = 1 und speichert eine Kopie vom Signalzustand des Datenbusses ab. Anschließend legt die Steuerzentrale die Adresse des Empfängerregisters auf den Adreßbus, und der Decoder bildet $e_2 = 1$. Mit $s_2 = 1$ und Takt 2 = 1 gibt der Zwischenspeicher RG N über Tor N die Daten wieder auf den Datenbus, so daß mit Takt 1 = 1 das Register 2 (G9) die Empfangserlaubnis EE = 1 erhält und die Daten vom Datenbus kopieren kann.

Diese Übertragung sieht wegen der Zwischenspeicherung im RG N etwas umständlich aus. Sie hat aber den Vorteil, daß die Adressenaktivierung von Sender und Empfänger nacheinander erfolgen kann und so Adressenzwischenspeicher und auch ein zweiter Decoder eingespart werden können.

Tristate-Gatter (Tristate-Tor). Damit die jeweils passiven Sendebausteine dem Bus keinen Strom entnehmen, hat man einen Gattertyp entwickelt, der einen dritten Zustand hat, in dem beide Ausgangstransistoren gesperrt sind. Ein solches Gatter nennt man ein *3-Zustands-Gatter* (Tristate-Gatter).

Bild 2.35 zeigt ein Tristate-NAND-Gatter in TTL-Technik. Die Erweiterungsschaltung (außerhalb des rot umrandeten Teils) funktioniert so, daß bei $s = 0$ T8 als Darlington-Vorverstärkerstufe für T3 wirkt und die normale NAND-Gatter-Funktion nicht beeinträchtigt. Anders ist das bei $s = 1$. Dann werden T6 und T7 leitend. Damit wird über E0 mit $e_0 = 0$ T1 leitend, T2 und T4 werden gesperrt. Die Basis-Emitter-Spannung an T8 wird so gering, daß T8 und damit T3 ebenfalls gesperrt werden.

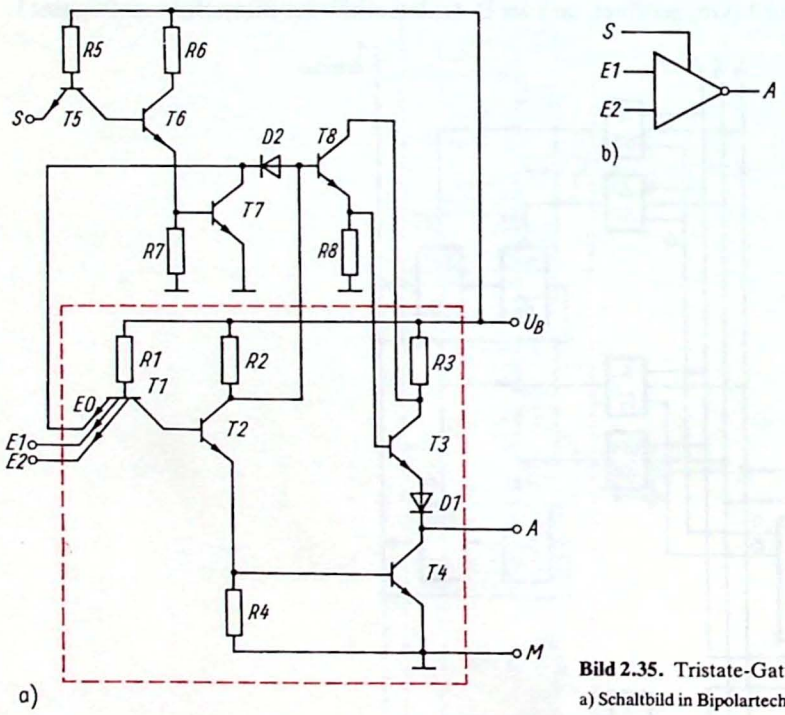


Bild 2.35. Tristate-Gatter
a) Schaltbild in Bipolartechnik; b) Schaltsymbol

Damit ist dann der dritte Zustand erreicht, bei dem sowohl T3 als auch T4 gesperrt sind und folglich der Ausgang A einen sehr hohen Widerstand aufweist. Es fließt dann kein Strom über A.

Verdrahtetes NOR. Ist das Tristate-Verhalten nicht unbedingt notwendig, kann man auch mit NAND-Gattern arbeiten, deren Endstufen keine Kollektorwiderstände enthalten (offene Kollektoren) **(Bild 2.36)**. Diese Variante hat den Vorteil, daß einfachere Koppelelemente verwendet werden, die dann aber auch keinen stromlosen dritten Zustand haben. Man nennt eine solche Kopplung ein *verdrahtetes* NOR (Wired-NOR).

Bei dieser Zusammenschaltung von mehreren Gattern mit ausgangsseitig offenem Kollektor wird der gemeinsame Kollektorwiderstand so ausgewählt, daß für die gewählte Anzahl von zusammengeschalteten Gattern optimale Bedingungen entstehen.
Bei beiden besprochenen Schaltungen, Tristate-Gatter und verdrahtetes NOR, ist zu beachten, daß die Ausgangsvariable y negiert ist.

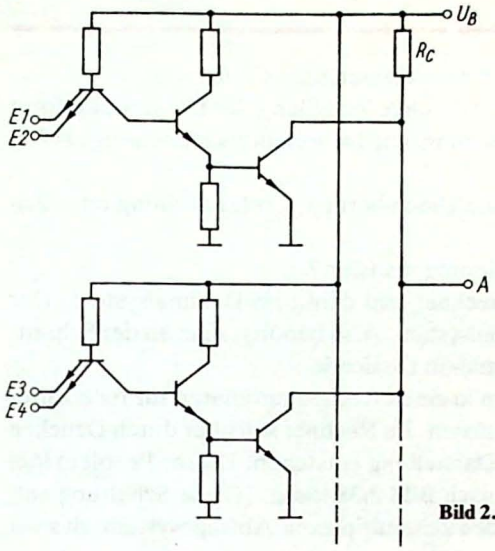


Bild 2.36. Verdrahtetes NOR

2.1.7.5. Vergleicher

Vergleicher benötigt man, um festzustellen, wann ein gesuchtes Wort in einer Wortfolge auftaucht.

Nehmen wir dazu das folgende Beispiel: An den Eingängen E11 bis E14 liegt ein vierstelliges Binärwort. An einem anderen Eingangsbündel E21 bis E24 sollen fortlaufend wechselnde, ebenfalls 4 Bit breite Binärwörter gegeben werden. Am Ausgang A der Schaltung soll dann der Zustand 1 erscheinen, wenn die an beiden Eingangsbündeln liegenden Binärwörter in jeder Bitstelle übereinstimmen. Die Lösung dieser Aufgabe kann mit vier Äquivalenzgliedern und einem UND-Glied realisiert werden **(Bild 2.37a)**.

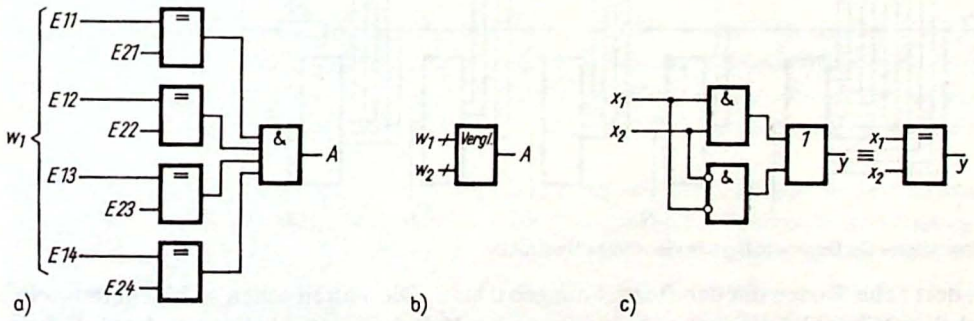


Bild 2.37. Vergleicher für zwei vierstellige Binärwörter
a) Signalflußbild; b) Schaltsymbol; c) Darstellung der Äquivalenzschaltung aus Grundschaltungsgliedern

Im **Bild 2.37c** ist die Äquivalenzschaltung aus noch einfacheren Grundschaltungsgliedern zusammengesetzt dargestellt. *Vielleicht überzeugen Sie sich mit Hilfe der Tafel 2.21, daß die angeführte identische Gleichheit berechtigt ist!*

2.1.7.6. Zuordner (Codierung, Umcodierung, Decodierung)

Wir wissen: Informationen lassen sich durch beliebige Zeichensysteme darstellen. Dabei muß im Fall verabredeter Zeichen eine eindeutige Zuordnungsvorschrift (Zustandstabelle, Codierungsvorschrift) bestehen.

Die Darstellung von Informationen nach einem Zeichensystem nennt man *Codierung*.
Die im Verlauf der Informationsübertragung oftmals notwendige Wandlung der Darstellungsform wird als *Umcodierung* und die Rückwandlung in die ursprüngliche Informationsdarstellung als *Decodierung* bezeichnet.
Allgemeiner betrachtet, stellt jeder dieser Prozesse eine Umcodierung, Codewandlung oder Zuordnung dar.
Wie werden nun Codierung, Umcodierung und Decodierung realisiert?
Nehmen wir dazu wieder ein Beispiel. Der Mensch rechnet und denkt im Dezimalsystem. Der Automat verarbeitet die gleichen Informationen im Dualsystem. Also benötigt man an der Schnittstellle Mensch–Automat ein Codewandlung von Dezimal- in Dualcode.
Beispielsweise möchten Sie bei der Eingabe von Zahlen in einen Rechenautomaten für jede Ziffer einer Dezimalzahl nur eine bestimmte Taste drücken müssen. Im Rechner soll aber durch Drücken dieser einen Taste die entsprechende Ziffer in dualer Darstellung entstehen. Dieses Problem läßt sich in vereinfachter Darstellung mit einer Schaltung nach **Bild 2.38** lösen. (Diese Schaltung entspricht nicht dem meist angewendeten leitungssparenden zeitmultiplexen Abfragesystem, das wir nicht besprechen wollen.)

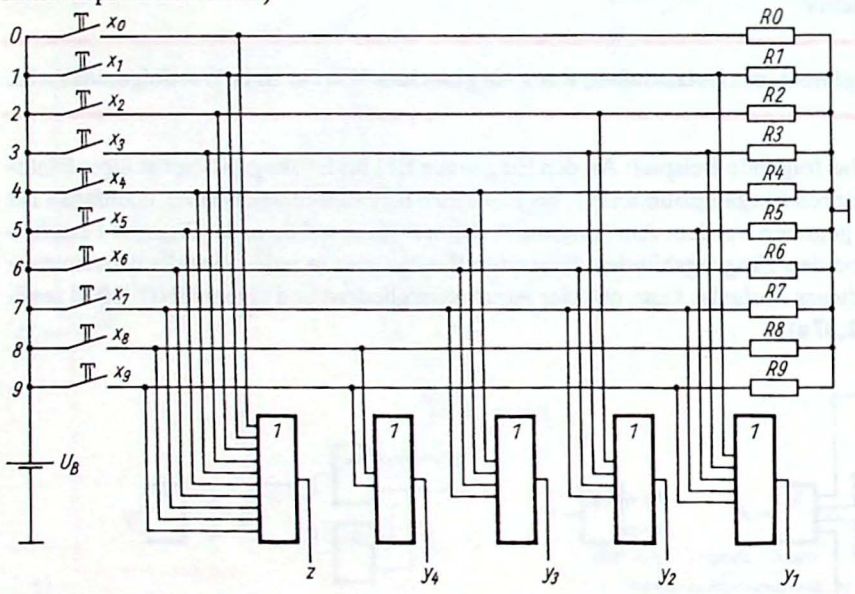


Bild 2.38. Umcodierer für Dezimalziffern in vierstellige Dualzahlen

Wir haben dort zehn Tasten mit den Bezeichnungen 0 bis 9. Die Tasten sollen so beschaffen sein, daß sie nach dem Niederdrücken von selbst wieder in ihre Ruhelage zurückkehren, in der sie keinen Kontakt geben. Wenn man eine Taste drückt, z. B. die Taste 6, so soll an den Ausgängen y_1 bis y_4 die dazugehörige Dualzahl erscheinen. Im Fall der Dezimalzahl 6 wird also:

$$y_4 = 0, \quad y_3 = 1, \quad y_2 = 1, \quad y_1 = 0.$$

Anhand der Schaltung können Sie sich überzeugen, daß dieser Sachverhalt eintritt.

Um übersehen zu können, welche Eingänge der ODER-Glieder mit welchen Tastenleitungen verbunden werden müssen, stellt man zweckmäßigerweise eine Codierungstabelle in Form der **Tafel 2.28** auf. In der ersten Spalte finden Sie dort die Angabe der Dezimalziffer. Die nächsten zehn Spalten zeigen, daß stets nur eine der Leitungen für die Variablen x_0 bis x_9 den Zustand 1 haben darf. In den letzten vier Spalten erkennen Sie zeilenweise die Dualzahlen von null bis neun. Wenn hier in einem Feld eine 1 auftritt, so bedeutet dies, daß das zu dieser Variablen y gehörende ODER-Glied mit einem Eingang an die zugehörige Tastenleitung geschaltet werden muß. Wird dann die zugehörige Tastenleitung in den Zustand 1 versetzt, so erscheint an dem Ausgang des ODER-Gliedes ebenfalls der Zustand 1.
Die Widerstände R0 bis R9 sind deswegen eingefügt, damit die Schaltung bei offenen Tasten niederohmig bleibt. Dadurch werden Störspannungen weniger wirksam.
Eine Kuriosität stellt der Nullzustand dar. Ist keine der Tasten 0 bis 9 gedrückt, liegt er bereits vor. Damit ist er eigentlich ein neutraler Zustand. Um nun der Schaltung mitzuteilen, daß dieser Zustand doch gewertet werden soll, muß die Nulltaste gedrückt werden. Dabei entsteht das Signal $z = 1$, das der der Codiereinrichtung nachfolgenden Schaltung die Übernahme des Nullzustands befiehlt. Auch beim Drücken der Tasten 1 bis 9 muß durch $z = 1$ die Übernahme der gebildeten Dualzahl angewiesen werden.

Tafel 2.28. Codierungstabelle für Bild 2.38

Dezimalzahl	Tastenvariable										Dualzahl			
	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_4	y_3	y_2	y_1
0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	**1**	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	**1**	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	**1**	0	0	0	0	0	0	0	0	1	1
4	0	0	0	0	**1**	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	**1**	0	0	0	0	0	1	0	1
6	0	0	0	0	0	0	**1**	0	0	0	0	1	1	0
7	0	0	0	0	0	0	0	**1**	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0	**1**	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	**1**	1	0	0	1

Die im Bild 2.38 dargestellte kombinatorische Schaltung kann auch als *Umcodierer* bezeichnet werden, der aus zehn jeweils einzeln besetzten Binärstellen eine Umcodierung in vier parallele Binärstellen realisiert.
Dieser Umcodierer kann noch einfacher aufgebaut werden, wenn man anstelle von Schaltkreisen für den Aufbau der ODER-Glieder Dioden und Widerstände verwendet **(Bild 2.39)**. Diese Dioden und Widerstände ergeben in ihrer Wirkung wieder ODER-Funktionen. Sie bieten aber, wie wir später sehen werden, den Übergang zu einer völlig neuen technischen und technologischen Konzeption.
Die Wirkung der Schaltung nach Bild 2.39 läßt sich wie folgt beschreiben: Wenn über eine Taste die Spannung U_B an eine Zeilenleitung gelegt wird, dann fließt über diejenigen Widerstände R1 bis R4 Strom, deren zugehörige Spaltenleitung über eine Diode mit der angeschalteten Zeilenleitung verbunden ist. Wird z. B. die Taste für x_5 gedrückt, dann kann Strom über R1 und R3 fließen. Die dadurch an R1 und R3 auftretenden positiven Spannungen ergeben den Zustand 1 für die Ausgangsvariablen y_1 und y_3. Die Spaltenleitungen, an denen R2 und R4 als Abschlußwiderstände liegen, führen bei $x_5 = 1$ keinen merklichen Strom, da über sie nur die Sperrströme einiger Dioden fließen können.
Beim Aufbau solcher Diodenmatrizen ist es also wichtig, Dioden zu verwenden, die in Sperrichtung einen sehr hohen Widerstand aufweisen und in Durchlaßrichtung hinreichend niederohmig sind.

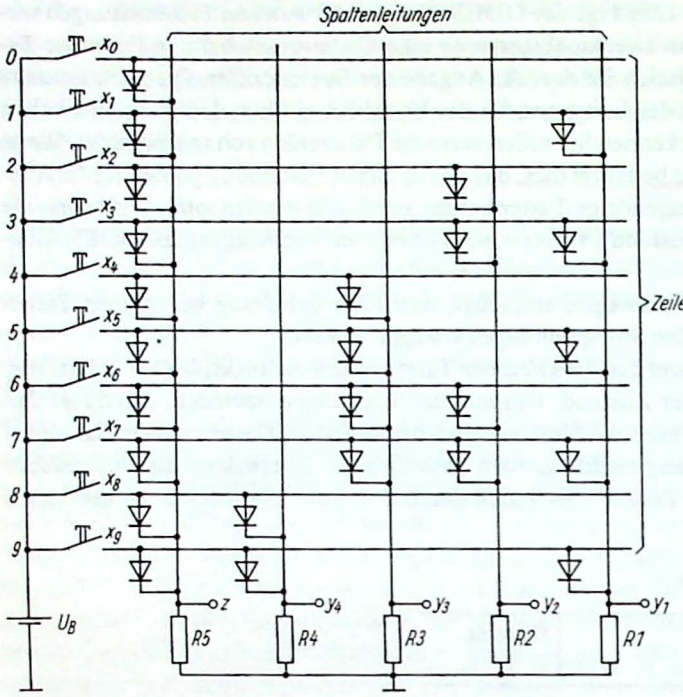


Bild 2.39
Umcodierer für Dezimalziffern in vierstellige Dualzahlen unter Verwendung einer Diodenmatrix

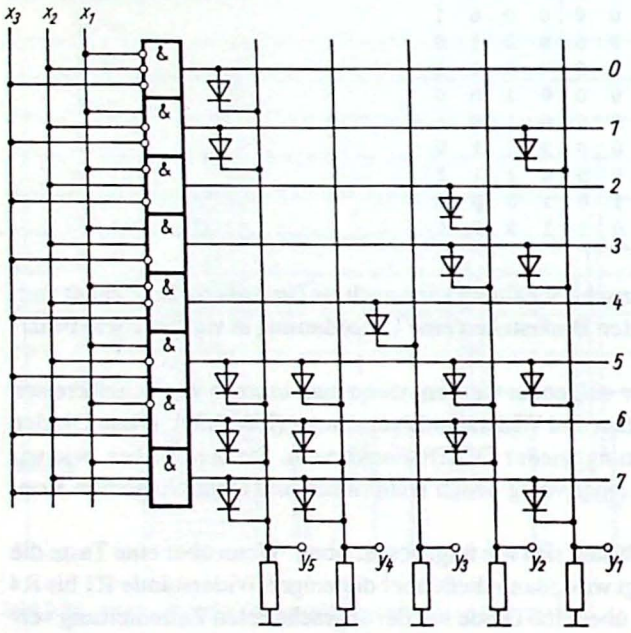


Bild 2.40
Umcodierer einer dreistelligen Dualzahl in eine fünfstellige Binärzeichenanordnung

Tafel 2.29. Codewandlungsvorschrift für Bild 2.40

x_3	x_2	x_1	y_5	y_4	y_3	y_2	y_1
0	0	0	1	0	0	0	0
0	0	1	1	0	0	0	1
0	1	0	0	0	0	1	0
0	1	1	0	0	0	1	1
1	0	0	0	0	1	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	0	1	0
1	1	1	1	1	0	0	1

Eine Umcodierung einer dreistelligen Dualzahl x_1, x_2, x_3 in eine fünfstellige Binärzeichenanordnung mit den Variablen y_1 bis y_5 wird im **Bild 2.40** und **Tafel 2.29** gezeigt. Die Codewandlungsvorschrift nach Tafel 2.29 ist dabei völlig willkürlich aufgestellt worden.

Als Beispiel für die Decodierung wollen wir die Rückwandlung von der Dualzahlen- in die Dezimalzahlendarstellung kennenlernen. **Bild 2.41** zeigt eine Decodierschaltung für vierstellige Dualzahlen in die Dezimalziffern Null bis Neun. Hierzu gehört die **Tafel 2.30**.

Beachten Sie bitte, daß die im Bild 2.41 dargestellte Schaltung Ähnlichkeit mit einem Demultiplexer hat. Es fehlt lediglich an jedem UND-Glied noch ein zusätzlicher Eingang für die durchzuschaltende Variable. Für diesen Vergleich müssen Sie die Variablen x_1 bis x_4 als Adreßvariable auffassen.

Als Erklärungsbeispiel für die Wirkungsweise dieser Decodierschaltung betrachten wir die Dualzahl 0111 B = 7. Mit $x_1 = 1, x_2 = 1, x_3 = 1$ und $x_4 = 0$ wird am UND-Glied mit der Ausgangsvariablen $y_7 = 1$ erscheinen.

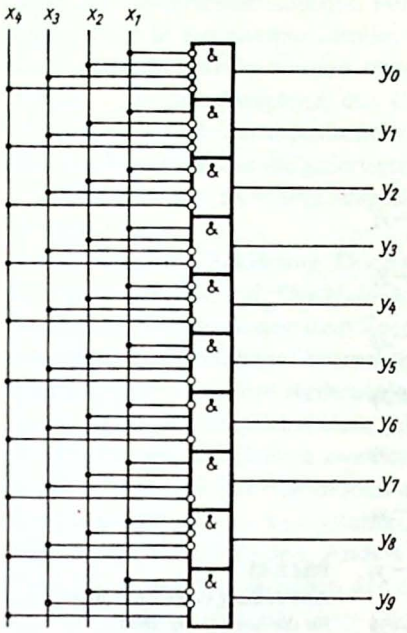


Bild 2.41
Decodierschaltung für vierstellige Dualzahlen in die Dezimalziffern Null bis Neun

Tafel 2.30. Codewandlungsvorschrift für Bild 2.41

x_4	x_3	x_2	x_1	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9
0	0	0	0	1									
0	0	0	1		1								
0	0	1	0			1							
0	0	1	1				1						
0	1	0	0					1					
0	1	0	1						1				
0	1	1	0							1			
0	1	1	1								1		
1	0	0	0									1	
1	0	0	1										1

Codierer, Umcodierer, Decodierer, zusammenfassend als Zuordner bezeichnet, haben also folgendes gemeinsam:

- Die möglichen Zeichen des Zeichensystems A (Eingabesystem) werden über UND-Glieder entschlüsselt, wodurch die zugehörige Zeilenleitung in den Zustand 1 gebracht wird.
  An diese Leitung sind ODER-Glieder dergestalt angeschlossen, daß an ihren Ausgängen die logischen Zustände des Zeichensystems B (Ausgabesystem) hervorgerufen werden.

Die in den Bildern 2.38, 2.39, 2.40, 2.41 dargestellten Signalflußbilder verdeutlichen diesen Sachverhalt. Im Sonderfall nach Bild 2.41 werden keine ODER-Glieder benötigt, da die durch das zugehörige UND-Glied identifizierte Leitung bereits den Variablenzustand darstellt.

Anwendungsbeispiel für einen Zuordner. Im folgenden wollen wir anhand eines kleinen Beispiels den Werdegang einer einfachen Informationsverarbeitung von der Informationsentstehung bis zur Umsetzung in die geplante Steuerwirkung kennenlernen. Wir werden sehen, daß Codierung, Umcodierung, Decodierung und die Wandlung der Energieform des Informationsträgers wesentliche Bestandteile dieses Prozesses sind.

Um bei der Massenfertigung von Präzisionsteilen eine wirtschaftliche Prozeßgestaltung zu erreichen, wendet man oftmals die Methode an, die Teile mit einer größeren Maßstreuung zu fertigen und später nach engeren Toleranzbereichen zu sortieren. Als Beispiel wollen wir die Zuordnung von Teilen gemäß ihrer bei der mechanischen Fertigung tatsächlich erzielten Länge zu bestimmten Toleranzgruppen wählen.

Bild 2.42 zeigt die wesentlichen Teile einer entsprechenden Informationsverarbeitungseinrichtung. Bei der Messung wird ein Codelineal (*1*) so verschoben, daß seine Position ein Maß für die Länge

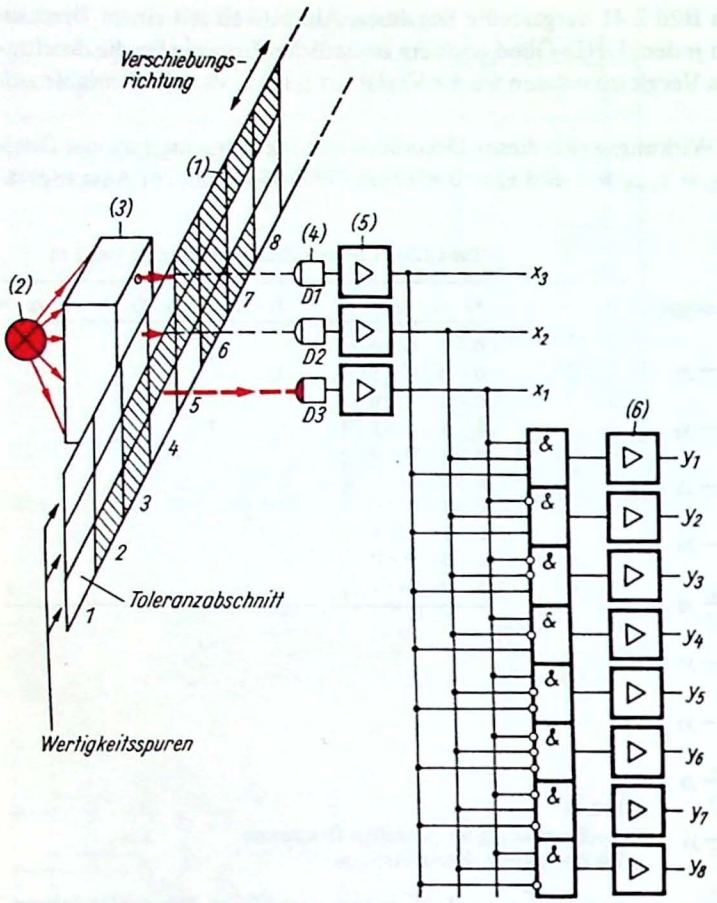


Bild 2.42
Anwendung eines Zuordners für die Sortierung nach Toleranzgruppen

Tafel 2.31. Zustandsdefinition für die Sortiereinrichtung nach Bild 2.42

Physikalischer Zustand	Symbolische Zuststand-kennzeichnung
Fotodiode *i* empfängt Licht	$x_i = 1$
Fotodiode *i* empfängt kein Licht	$x_i = 0$
Am Ausgang *i* tritt Stromfluß auf	$y_i = 1$
Am Ausgang *i* tritt kein Stromfluß auf	$y_i = 0$

Tafel 2.32. Codewandlungsvorschrift für den Umcodierer der Sortiereinrichtung nach Bild 2.42

Toleranz-bereich	Codierungs-zustand			Zustand der Ausgangsströme							
	x_3	x_2	x_1	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
1	1	1	1	1							
2	1	1	0		1						
3	1	0	0			1					
4	1	0	1				1				
5	0	0	1					1			
6	0	0	0						1		
7	0	1	0							1	
8	0	1	1								1

des Prüflings ergibt. Das Codelineal ist in senkrechte Streifen unterteilt, deren Breite ein Maß für den vorgegebenen Toleranzbereich darstellen soll. Insgesamt existieren acht solcher Toleranzbereiche, die im Bild 2.42 oder in **Tafel 2.32** von eins bis acht durchnumeriert sind. Jeder einzelne Streifen weist ein bestimmtes Muster von lichtdurchlässigen und lichtundurchlässigen Flächen auf. Dieses Muster dient zur Umwandlung der Meßinformation in ein binärcodiertes Signal. Die Strahlen der Lichtquelle (*2*) werden über ein optisches System (*3*) so gebündelt, daß drei Einzelstrahlen entstehen, deren Durchmesser etwa ein Zehntel der Toleranzfeldbreite beträgt. Auf der anderen Seite des Codelineals sind drei Fotodioden (*4*) angebracht, die die jeweils empfangene Beleuchtungsstärke in ein entsprechendes elektrisches Binärsignal **(Tafel 2.31)** umwandeln. Die elektrischen Signalzustände werden über Verstärker (*5*) verstärkt und einem Umcodiernetzwerk zugeführt. An die Ausgänge der Codiereinrichtung sind weitere Verstärker (*6*) geschaltet, die entsprechende elektromechanische Stelleinrichtungen betätigen. Das können in unserem Beispiel Zugmagnete sein, die die gefertigten Teile in zugehörige Sortierfächer lenken. Es können auch die Typenhebel eines Druckers ausgelöst werden, die einen bestimmten Kennbuchstaben auf das Teil drucken.

Soweit die grobe Erklärung. Doch sehen wir uns noch einige Einzelheiten an. Das Codelineal weist eine Besonderheit auf. Die Hell-Dunkel-Verteilung auf den Streifen stellt ein binäres Muster dar. Die hellen Flächen sollen dem Zustand 1, die dunklen dem Zustand 0 entsprechen. Wenn Sie sich daraufhin das Codelineal betrachten, werden Sie die Muster aller Dualzahlen von null bis sieben wiederfinden, aber ihre Reihenfolge ist nicht die übliche. Der Grund dafür liegt darin, daß die Irrtumsmöglichkeit möglichst klein gehalten werden soll. Sie ist dadurch gegeben, daß die Lichtstrahlen gerade auf der Grenze zwischen zwei Toleranzstreifen stehen können. Wegen der nie völlig exakten Montage der Fotodioden und des optischen Systems können dann alle möglichen Wertekombinationen x_1, x_2, x_3 entstehen, die durch die eine oder andere Möglichkeit der benachbarten Flächen entstehen können. Ändert man jeweils den Helligkeitswert nur auf einer Wertigkeitsspur, kann ein Irrtum nur zwischen den beiden benachbarten Werten eintreten. Diese irrtumsminimale Codierungsform wird als *Gray-Code* bezeichnet. Abschließend sollen noch einmal alle Informationsverarbeitungsstufen der besprochenen Sortiereinrichtung aufgeführt werden:

- Messen der Länge
- Wandlung der Länge in eine entsprechende Stellung des Codelineals
- Codierung des zur gemessenen Länge gehörenden Toleranzbereichs nach dem Gray-Code und Übertragung dieser Information auf die Variablen x_1, x_2, x_3
- Umwandlung des Zustands der Variablen x_i in den Zustand der Variablen y_i (Umcodierung oder Decodierung $x_1, x_2, x_3 \rightarrow y_i$)
- Auswertung von y_i mit Hilfe einer Stelleinrichtung.

Vielleicht entsteht nun bei Ihnen die Frage, ob man nicht die Stellung des Lineals ohne Codierung nach *Gray* durch acht Lichtstrahldurchtrittspunkte gleich in den zugehörigen y-Wert umsetzen könnte. Diese Lösung erfordert aber einen Mehraufwand von fünf Fotodioden mit den jeweils zugehörigen Verstärkern und von fünf Übertragungsleitungen. Außerdem wäre noch das Problem der Doppeldeutigkeit an der Übergangsstelle von einem Abschnitt zum anderen zu lösen. Bei noch mehr als acht Sortiergruppen ist auf alle Fälle die im Bild 2.42 gezeigte Form der Informationsverarbeitung günstiger.

Universeller Zuordner (Umcodierer). Mit Hilfe der zur Decodierung der Eingangswörter benötigten UND-Glieder und einer nachträglich mit Dioden bestückbaren Matrix läßt sich vorteilhaft ein Umcodierer mit veränderbarer Zuordnungstabelle aufbauen. Man braucht dann nur noch die Dioden in Abhängigkeit von der vorgegebenen Umcodierungsvorschrift einzufügen.

Einen solchen universellen Zuordner (Umcodierer) zeigt **Bild 2.43**. Die Dioden sind alle weggelassen, damit die variable Positionierungsmöglichkeit angedeutet ist. Er ist als variabler Umcodierer

für eine 4-Bit-Eingangscode auf einen 4-Bit-Ausgangscode ausgelegt. Die Zuordnung der Zeilenleitungen für die Eingangsvariablen ist nach dem Dualzahlenschema vorgenommen worden. Interessant ist, daß man den universellen Zuordner, wenn er dann fertig mit Dioden bestückt ist, auch noch ganz anders betrachten kann:

- Man kann ihn als *Festwertspeicher* ansehen.

Er hat nämlich die Eigenschaft, auf eine bestimmte Eingangsdualzahl eine bestimmte Bitkombination am Ausgang zu erzeugen, und zwar immer die gleiche. Er hat sich also fest gemerkt, welche Ausgangsbitkombination zu welcher Dualzahl am Eingang gehört.

- Speicher dieser Art werden als Festwertspeicher oder als Nur-Lese-Speicher oder als ROM (read only memory) bezeichnet.

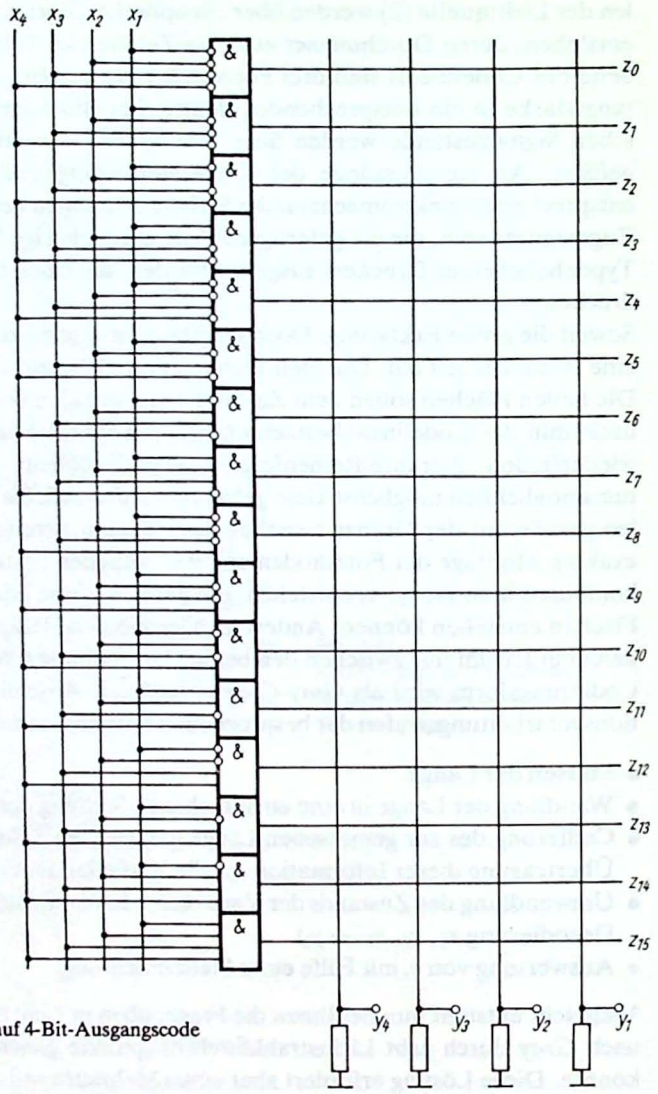


Bild 2.43
Variabler Umcodierer für 4-Bit-Eingangscode auf 4-Bit-Ausgangscode

2.1.7.7. Festwertspeicher: ROM

Der ROM (read only memory) ist ein Festwertspeicher. Auf jede der möglichen Eingangsbitkombinationen erzeugt er eine vom Anwender vorgebbare Bitkombination am Ausgang.
Der ROM ist demnach nichts weiter als ein universeller Umcodierer. Die Anzahl seiner Eingangs- und Ausgangsvariablen wird der Aufgabenstellung angepaßt.
Der ROM ist vorrangig dafür entwickelt worden, daß die in ihm gespeicherten Bitkombinationen der Reihe nach abgerufen werden und in dieser Reihenfolge die beabsichtigten Steuereffekte wirksam werden.

Zur Demonstration des Prinzips wollen wir mit einem einfachen Beispiel nach **Bild 2.44** beginnen und zunächst noch mit Kontakten schalten.
Ist im Bild 2.44 Kontakt S1 geschlossen, so lautet die Ausgangsbitkombination:

$$y_1 = 1, \quad y_2 = 1, \quad y_3 = 1.$$

Ist S2 geschlossen, erhält man am Ausgang:

$$y_1 = 1, \quad y_2 = 1, \quad y_3 = 0.$$

Bei S3 geschlossen folgt:

$$y_1 = 1, \quad y_2 = 0, \quad y_3 = 1.$$

Tafel 2.33 zeigt noch einmal eine Zusammenstellung der drei möglichen Zustände, wobei x_i die Zustandsvariablen der Schalter S_i bezeichnet.

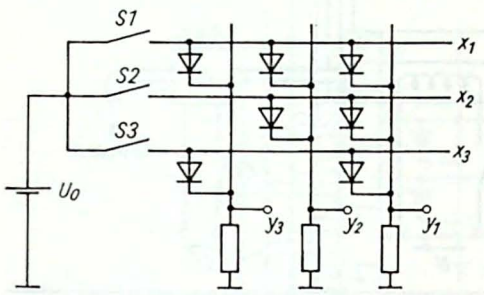


Bild 2.44
Einfache ROM-Schaltung

Tafel 2.33. Zuordungsvorschrift für den ROM nach Bild 2.44

x_1	x_2	x_3	y_3	y_2	y_1
1			1	1	1
	1		0	1	1
		1	1	0	1

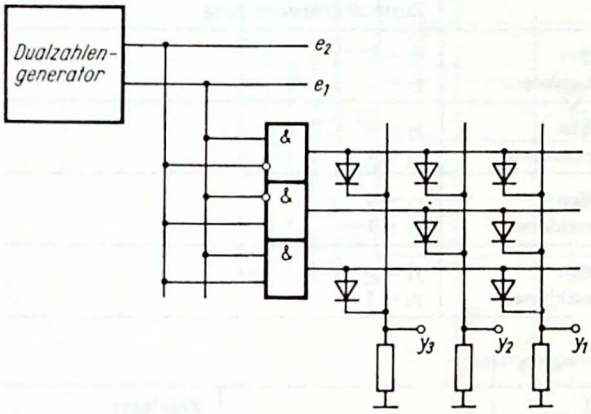


Bild 2.45
Zyklische Ansteuerung eines einfachen ROM durch Dualzahlengenerator und Decoder

Tafel 2.34. Zuordungsvorschrift für die ROM-Abfragung nach Bild 2.45

e_2	e_1	y_3	y_2	y_1
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	1	0	1

Um diese drei Zustände der Reihe nach zu erzeugen, kann man mit Taktgenerator und Zähler einen *Dualzahlengenerator* (Näheres später auf S. 120) aufbauen und über eine Decodiereinrichtung die Zeilen des ROM der Reihe nach in den Zustand 1 bringen **(Bild 2.45, Tafel 2.34)**.
Mit jedem Taktimpuls wird der Zähler um eine Stelle weitergestellt, erzeugt also die nachfolgende Dualzahl. Mit jeder neuen Dualzahl wird die «ROM-Abfragung» um eine Zeile weitergeschaltet. Mit jeder Zeile entsteht eine bestimmte, vorausgeplante Bitbelegung an den Ausgängen, mit der die beabsichtigten Steuerwirkungen realisiert werden.
Schauen wir uns dazu ein einfaches Beispiel an. Nehmen wir an, daß wir den Prozeß einer Waschmaschine nach einem vorgegebenen Zeitplan **(Tafel 2.35)** zu steuern haben. Im **Bild 2.46** sind alle zu steuernden Teile der Waschmaschine in einer Prinzipskizze dargestellt. Der abwechselnde Rechts-Links-Lauf des Motors wird aus Vereinfachungsgründen nicht beachtet. Um zu einem Zustandssystem zu gelangen, führen wir die in **Tafel 2.36** zusammengestellten Definitionen ein.

Tafel 2.35
Zeitplan für ein Waschmaschinenprogramm

Nr.	Prozeßschritt	Dauer
1	Wasser einlassen	20 s
2	Heizen, Trommel laufen lassen	50 min
3	Heizung aus, Trommel läuft	20 min
4	Wasser ablassen	30 s
5	Wasser einlassen	20 s
6	Trommel laufen lassen	3 min
7	Wasser ablassen	30 s
8	Wasser einlassen	20 s
9	Trommel laufen lassen	3 min
10	Wasser ablassen	30 s

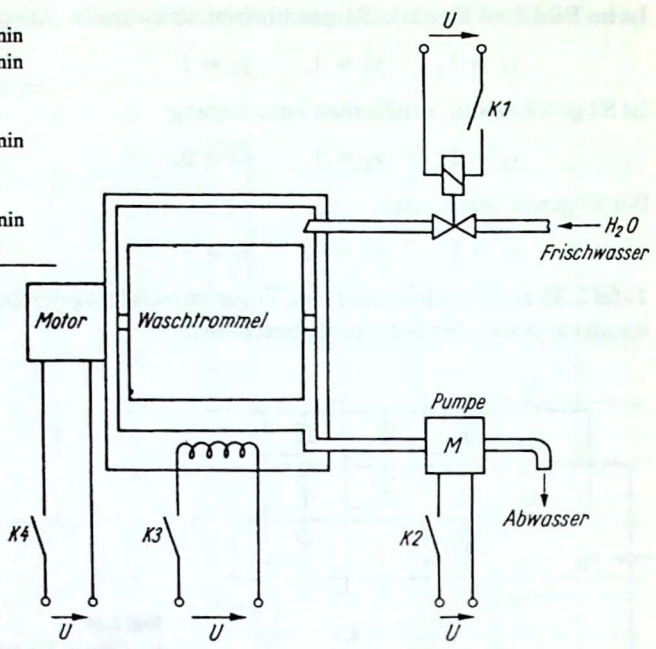


Bild 2.46
Prinzipskizze der zu steuernden Teile einer Waschmaschine

Tafel 2.36. Zustandsdefinitionen für die Waschmaschinensteuerung nach Bild 2.46

Physikalischer Zustand	Kontaktstellung	Symbolische Zustandskennzeichnung
Frischwasserzufuhr gesperrt	K1 offen	$y_1 = 0$
Frischwasser wird zugeführt	K1 geschlossen	$y_1 = 1$
Abwasserpumpe ausgeschaltet	K2 offen	$y_2 = 0$
Abwasserpumpe arbeitet	K2 geschlossen	$y_2 = 1$
Heizung abgestellt	K3 offen	$y_3 = 0$
Heizung in Betrieb	K3 geschlossen	$y_3 = 1$
Motor steht still	K4 offen	$y_4 = 0$
Motor dreht sich	K4 geschlossen	$y_4 = 1$

Tafel 2.37. Zuordnungsvorschrift für das Waschmaschinenprogramm

Zustandszahl dezimal	dual e_4	e_3	e_2	e_1	Zustand Motor y_4	Heizung y_3	Abwasser y_2	Frischwasser y_1	Zeitdauer
0	0	0	0	0	0	0	0	0	Stillstand
1	0	0	0	1	0	0	0	1	20 s
2	0	0	1	0	1	1	0	0	50 min
3	0	0	1	1	1	0	0	0	20 min
4	0	1	0	0	0	0	1	0	30 s
5	0	1	0	1	0	0	0	1	20 s
6	0	1	1	0	1	0	0	0	3 min
7	0	1	1	1	0	0	1	0	30 s
8	1	0	0	0	0	0	0	1	20 s
9	1	0	0	1	1	0	0	0	3 min
10	1	0	1	0	0	0	1	0	30 s

Aus den Tafeln 2.35 und 2.36 läßt sich die in **Tafel 2.37** dargestellte Zustandsfolge ableiten. Zur technischen Realisierung soll ein ROM verwendet werden, in dem die Zustände der y-Werte der Reihe nach gespeichert werden **(Bild 2.47)**. Dabei wird die in Tafel 2.37 rot umrandete Tabelle sinngemäß im ROM abgebildet. Dort, wo in der Tabelle eine 1 steht, wird im ROM eine Diode eingesetzt. Außerdem benötigt man noch einen Zeitgeber, der dem ebenfalls benötigten Zähler durch einen elektrischen Impuls mitteilt, wann er seinen Zählzustand um eins erhöhen soll. Des weiteren muß noch eine Decodiereinrichtung für die Dualzahlen von null bis zehn vorhanden sein. Die im Bild 2.47 an den Widerständen R1 bis R4 auftretenden Spannungen entsprechen den Zuständen der Variablen y_1 bis y_4. Nach entsprechender Verstärkung lassen diese Spannungen die zugehörigen Relais ansprechen, und der jeweilige Steuerungsvorgang wird ausgelöst.

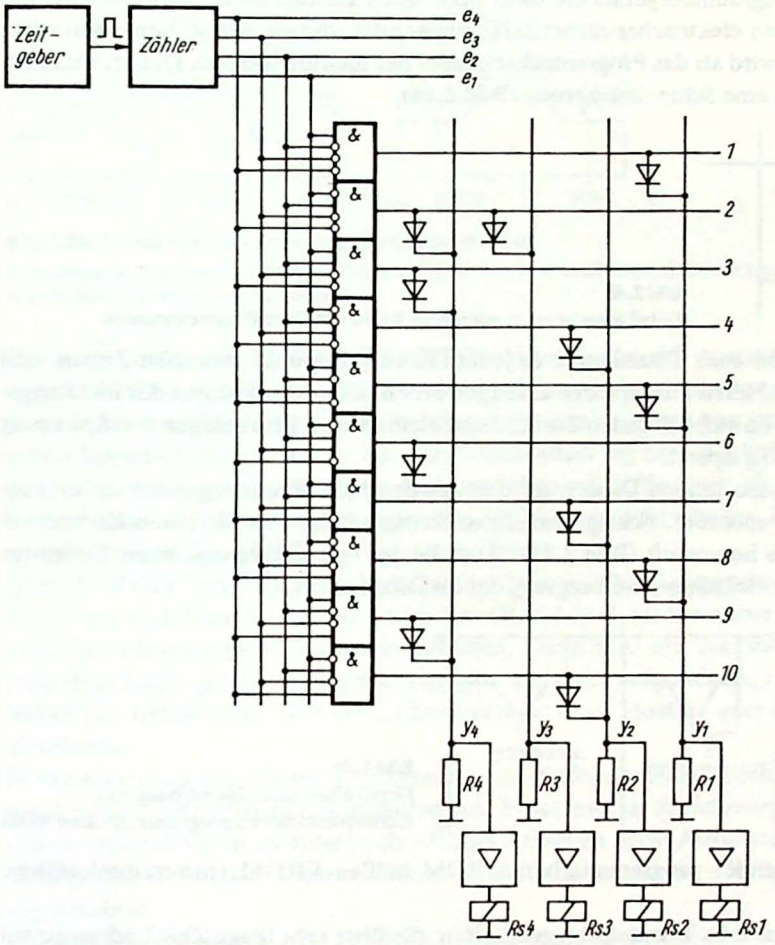


Bild 2.47. ROM-Steuerung zur Realisierung des Waschmaschinen-Programms

Natürlich können jetzt Kritiker behaupten, daß das alles recht umständlich und aufwendig ist, aber

- man kann den ROM leicht mit einem anderen Programm versehen
- man kann mit mikroelektronischen Bauelementen alles sehr klein gestalten.

Außerdem sollte noch beachtet werden, daß das Beispiel die Aufgabe hat, die ROM-Anwendung zu erklären.

Wie sehen nun die ROM der Mikroelektronik aus, wie werden sie gefertigt, und wie werden sie eingesetzt?

ROM-Strukturen werden auf Halbleiterchips aufgebracht, die in der Größenordnung von 5 mm × 5 mm liegen. Auf dieser kleinen Fläche kann man über 10^5 Bit an Speicherstellen einschließlich der zugehörigen Decodiereinrichtungen, Multiplexer und Tristate-Gatter unterbringen.
Die Dioden können an den gewünschten Leitungskreuzungen auf verschiedene Art eingefügt werden. Benötigt man für größere Fertigungsserien (Stückzahl 10^2 und größer) Festwertspeicher, die alle das gleiche Programm aufweisen, so kann man den ROM fertig programmiert vom Hersteller beziehen. Die gewünschten Dioden werden mit *hineingefertigt*. Das ist der gleiche Prozeß wie bei der Herstellung anderer Schaltkreise (Maskenprogrammierung).
Das andere Konzept (für geringere Stückzahlen) sieht vor, daß der Hersteller an jeder Kreuzung zwischen Zeilen- und Spaltenleitung eine Diode hineinfertigt. Der Anwender hat dann die Möglichkeit, mit Hilfe eines Programmiergeräts die nicht benötigten Dioden unwirksam zu machen. Dazu wird eine Überdosis an elektrischer Stromstärke verwendet, die gerade so bemessen wird, daß nichts anderes zerstört wird als das Programmierelement der nichtbenötigten Diode. Das Programmierelement wirkt wie eine Schmelzsicherung **(Bild 2.48)**.

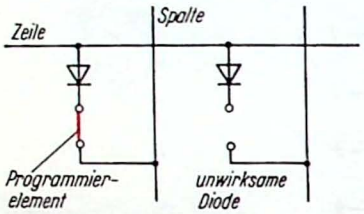


Bild 2.48
Detail eines programmierbaren ROM mit Durchbrennelementen

Anstelle der Dioden können auch Transistoren in jedem Kreuzungspunkt zwischen Zeilen- und Spaltenleitungen durch den Herstellungsprozeß erzeugt werden. Diese sind wegen der nicht angeschlossenen Basiselektrode im nichtleitenden Zustand und bleiben auch bei Anlegen von Spannung zwischen Zeilen und Spalten gesperrt.
Aus den Transistoren können jedoch Dioden an den gewünschten Kreuzungspunkten erzeugt werden, indem man durch mehrere, richtig bemessene Stromimpulse jeweils einen Kurzschluß zwischen Emitter und Basis hervorruft **(Bild 2.49)**. Anstelle des npn-Übergangs beim Transistor verbleibt dann nur noch ein einfacher pn-Übergang, der als Diode wirkt.

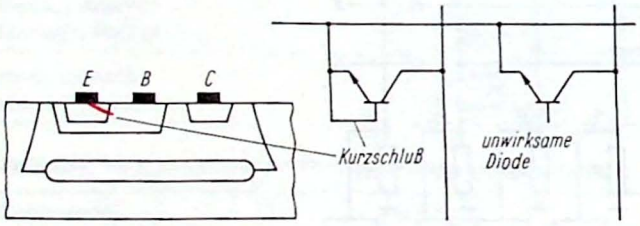


Bild 2.49
Detail eines durch Herstellung von Kurzschlußbrücken programmierbaren ROM

Diese einmalig vom Anwender programmierbaren ROM heißen PROM (programmierfähige ROM).
In der MOS-Technologie hat man Transistoren geschaffen, die über sehr lange Zeit Ladungen auf einem zusätzlichen, im Isoliermaterial potentialmäßig „schwimmenden" Gate (floating gate) speichern können (Bilder 2.50a, b).
Beim EEPROM **(Bild 2.50a)** können diese Ladungen durch entsprechend gepolte höhere Spannungen infolge der sehr dünnen Gate-Oxid-Schicht sowohl auf das Floating-gate heraufgebracht als auch wieder heruntergeholt werden. Dadurch nimmt der Kanal zwischen Source und Drain jeweils einen anderen Leitfähigkeitszustand an. In der Speicherzelle **(Bild 2.50c)** ist der Floating-gate-Transistor noch mit einem Auswahltransistor gekoppelt. Durch Aufsteuern dieses Auswahltransistors mit Hilfe der Decodierlogik erhält nun je nach dem Ladungszustand des Floating-gate die Spaltenleitung den Zustand 0 oder 1.

Das Laden und Entladen des Floating-gate kann beim EEPROM in der Einsatzschaltung des Speichers erfolgen. Nachteilig ist nur, daß dazu eine spezielle Steuerschaltung benötigt wird. Diese kann aber mit in den Speicherschaltkreis integriert werden. Die maximal zulässige Anzahl der Umspeicherungen ist begrenzt. Man gibt sie mit 10^4 bis 10^6 an. Vorteilhaft ist die Möglichkeit, jede Zelle einzeln umspeichern zu können. Die Umspeicherzeit (Schreibzeit) beträgt 1 bis 20 ms je Byte.

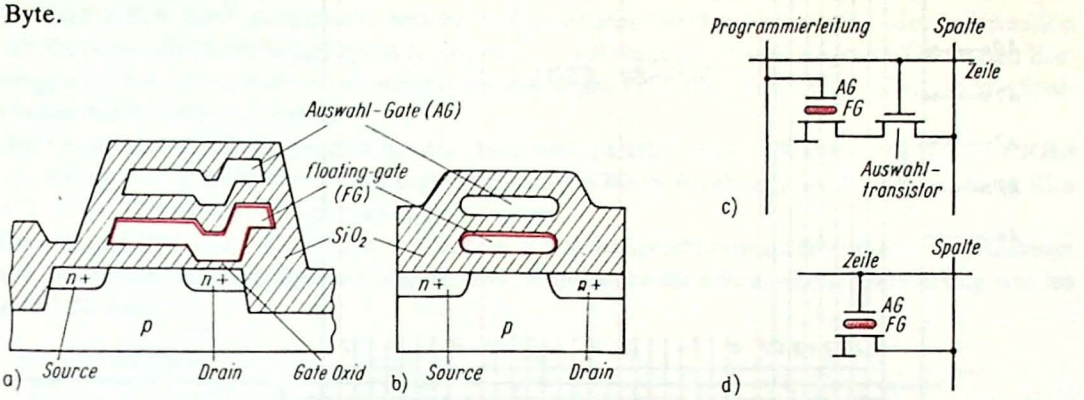


Bild 2.50. Detail von lösch- und programmierbaren ROM

a) Floating-gate-Transistor für EEPROM; b) Floating-gate-Transistor für EPROM; c) einzelne Speicherzelle mit Auswahltransistor; d) einzelne Speicherzelle ohne Auswahltransistor

Der länger bekannte EPROM **(Bild 2.50b)** unterscheidet sich in seinem Aufbau nur unwesentlich vom EEPROM. Infolge der dickeren Oxidschicht zwischen Floating-gate und Drain-source-Kanal können bei ihm die Ladungen elektrisch nur in einer Richtung transportiert werden. Zum Ändern seines Speicherinhalts muß er aus der Einsatzschaltung herausgeholt werden. Gelöscht wird er dann über ein eingebautes Quarzglasfensterchen mit Hilfe einer 10- bis 20minütigen UV-Licht-Bestrahlung. Dabei kann nur der gesamte Speicher gelöscht werden. Zum Programmieren kommt er auf ein spezielles EPROM-Programmiergerät.

Beim EPROM kann die Speicherzelle ebenfalls aus der Zusammenschaltung von Floating-gate-Transistor und Auswahltransistor bestehen (Bild 2.50c). Sie kann aber auch, wie im **Bild 2.50d**, nur noch den Floating-gate-Transistor enthalten. Dann wird der Transistor bei Anwahl der Zeile je nach dem Ladungszustand des Floating-gate entweder aufgesteuert, oder er bleibt gesperrt. Nach bisherigen Erfahrungen soll der Ladungszustand eines Floating-gate in 10 Jahren höchstens 30 % abnehmen.

Damit haben wir eine kleine Übersicht über gebräuchliche ROM-Technologien erhalten. Anschließend müssen wir noch einige grundlegende Probleme zur *Speicherorganisation* kennenlernen. Es wurde schon erwähnt, daß mehr als 10^5 Speicherzellen nebst Auswertelogik auf einem ROM-Chip untergebracht werden können. Diese hohe Elementeanzahl erfordert eine effektive Speicherorganisation.

Würde man ROM mit z. B. 10^3 Speicherzellen und mehr dergestalt organisieren, wie wir es im Bild2.47 getan haben, so würden unförmig lange, schmale Speichergebilde entstehen. Man strebt aber ein quadratisches Chipformat an. Außerdem paßt man die Speicherkapazität an die Dualzahlpotenzen und an die gebräuchlichen Wortbreiten an.

So gibt es z. B. ROM mit 32 Zeilen und 32 Spalten. Eine gebräuchliche Organisationsform eines solchen ROM ist im **Bild 2.51a** dargestellt.

Sie erkennen, daß das Ausgabewort an den Datenausgängen D1 bis D4 erscheint, also 4 Bit breit ist. Die Wertebelegung für jedes Bit dieses Ausgabeworts wird über einen zugehörigen 8 : 1-Multiplexer aus dem Zustand einer von acht Spaltenleitungen gebildet. Damit werden folgende Adreßinformationen benötigt: 3 Bit für die an den Adreßeingängen (A7 bis A5) parallel geschalteten

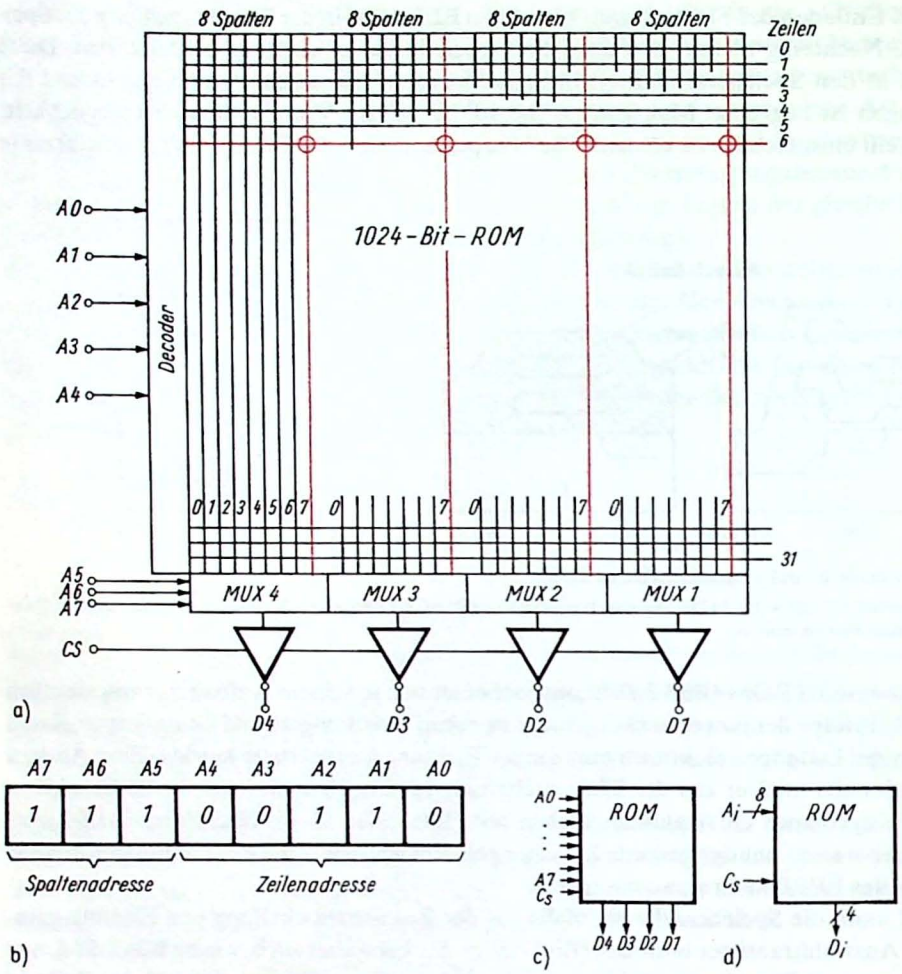
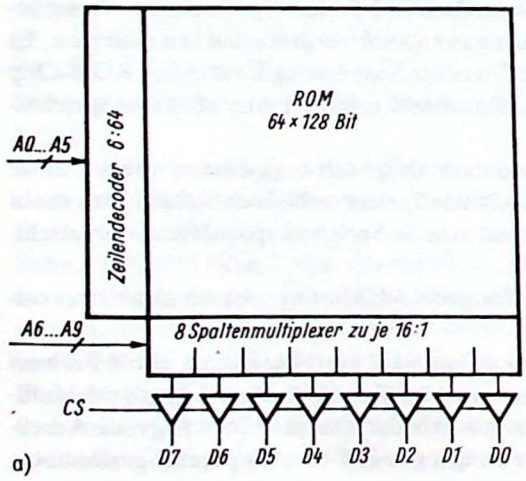


Bild 2.51. 1024-Bit-ROM zur Speicherung von 256 4-Bit-Wörtern

a) schematische Darstellung; b) Adressenbeispiel; c) Schaltsymbol mit Einzelleitungen; d) vereinfachtes Schaltsymbol mit Leitungsbündeln



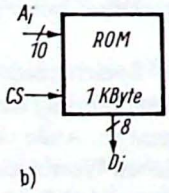


Bild 2.52. 1 K × 8 Bit-ROM

a) schematische Darstellung; b) vereinfachtes Schaltsymbol

Multiplexer und 5 Bit an den Eingängen A4 bis A0 für die Auswahl von jeweils einer aus 32 Zeilenleitungen. Es können also 8 × 32 = 256 verschiedene 4-Bit-Ausgangswörter aus dem Speicher abgerufen werden. Mit Hilfe des Anschlusses CS (chip select ≙ Chipauswahl) kann man in Abhängigkeit vom logischen Zustand an CS den Inhalt der angewählten ROM-Speicherzellen ausgeben oder zurückhalten (Tristate-Gatter).

Im **Bild 2.51b** wird dargestellt, welche Zellen gelesen werden, wenn die Adreßinformation 111001 10 für die Adreßeingänge A7, A6, A5, ..., A0 vorliegt. Es wird wegen 00110 an den Eingängen A4 bis A0 die Zeile sechs ausgewählt und wegen 111 an den Eingängen A7 bis A5 alle Spalten mit der Bezifferung sieben.

Der beschriebene ROM-Speicher hat eine Speicherkapazität von 32 × 32 Bit = 1024 Bit = 1 KBit = $\frac{1}{8}$ KByte. Zu 1024 Bit = 2^{10} Bit sagt man auch 1 KBit in Anlehnung an die Bezeichnung Kilo = k = 10^3. Zu 1024 Byte sagt man analog 1 KByte.

Bei einem ROM von 1 K × 8 Bit = 1 KByte = 8 KBit Speicherkapazität wird eine Organisation von 64 Zeilen und 128 Spalten angewendet. Dann sieht die schematische Darstellung wie im **Bild 2.52** aus.

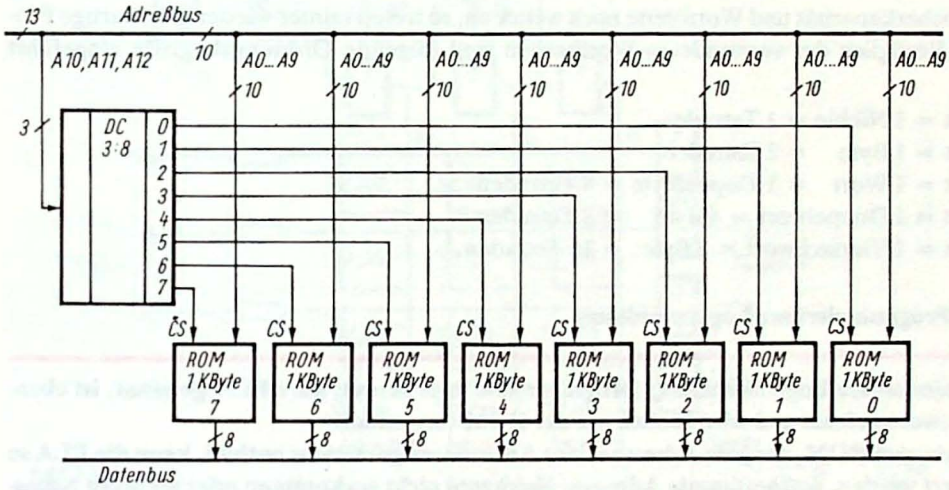


Bild 2.53. Speicher mit einer Kapazität von 8 KByte

Will man mit diesem 1-KByte-ROM z. B. eine Gesamtspeicherkapazität von 8 KByte zusammenstellen, so benötigt man 8 Speicherschaltkreise und einen 3 : 8-Decoderbaustein zur Adressierungsauswahl jedes einzelnen ROM **(Bild 2.53)**. Die einzelnen Speicherbausteine werden gemeinsam durch die Adreßbits A0 bis A9 angewählt. Die Adreßbits A10, A11, A12 entscheiden jedoch über den Eingang CS, welcher Baustein seine Ausgangsdaten über die Tristate-Tore auf den Datenbus legen kann. So hat z. B. der Baustein 0 den Adreßbereich von 0000 H bis 03FF H, der Baustein 1 den Adreßbereich 0400 H bis 07FF H usw. (siehe dazu **Bild 2.54)**.

Vielleicht überprüfen Sie sich jetzt, ob Sie folgende Aufgabe lösen können:

ROM-Speicherbausteine vom Typ 8 K × 8 Bit = 8 KByte = 64 KBit sollen mit Hilfe eines 3 : 8-Decoders so zusammengeschaltet werden, daß eine Gesamtspeicherkapazität von 64 KByte entsteht. Wie lauten die Adressenbereiche der einzelnen Bausteine im Hexadezimalcode?

Ergebnis: 0000 H – 1FFF H; 2000 H – 3FFF H; 4000 H – 5FFF H;
6000 H – 7FFF H; 8000 H – 9FFF H; A000 H – BFFF H;
C000 H – DFFF H; E000 H – FFFF H.

a)

Baustein Nr.	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
		Tetrade 2				Tetrade 1				Tetrade 0			
0	0	0	0	X	X	X	X	X	X	X	X	X	X
1	0	0	1	X	X	X	X	X	X	X	X	X	X
2	0	1	0	X	X	X	X	X	X	X	X	X	X
3	0	1	1	X	X	X	X	X	X	X	X	X	X
4	1	0	0	X	X	X	X	X	X	X	X	X	X
5	1	0	1	X	X	X	X	X	X	X	X	X	X
6	1	1	0	X	X	X	X	X	X	X	X	X	X
7	1	1	1	X	X	X	X	X	X	X	X	X	X

b)

0000H – 03FFH
0400H – 07FFH
0800H – 0BFFH
0C00H – 0FFFH
1000H – 13FFH
1400H – 17FFH
1800H – 1BFFH
1C00H – 1FFFH

Bild 2.54. Adressenbereiche der einzelnen Speicherbausteine nach Bild 2.53
a) in binärer Form (× ≙ wahlweise Belegung mit 0 oder 1); b) in hexadezimaler Form

Steigen Speicherkapazität und Wortbreite noch weiter an, so treten immer wieder gleichartige Probleme auf. Bezüglich der verwendeten Wortbreiten sind folgende Ordnungsbegriffe eingeführt worden:

4-Bit-Wort = 1 Nibble = 1 Tetrade
8-Bit-Wort = 1 Byte = 2 Tetraden
16-Bit-Wort = 1 Wort = 1 Doppelbyte = 4 Tetraden
32-Bit-Wort = 1 Doppelwort = 4 Byte = 8 Tetraden
64-Bit-Wort = 1 Vierfachwort = 8 Byte = 16 Tetraden.

2.1.7.8. Programmierbare Logikanordnung

Die programmierbare Logikanordnung (programmable logic array), kurz PLA genannt, ist ebenfalls ein Festwertspeicher und wird ähnlich wie der ROM verwendet.
Im Gegensatz zum ROM, der jede Adresse seines Adressierungsbereichs enthält, kann die PLA so programmiert werden, daß bestimmte Adressen überhaupt nicht vorkommen oder mehrere Adressen auf ein und dieselbe Bitkombination am Ausgang führen.
Die PLA läßt eine Optimierung bezüglich spezieller Anwendungsfälle zu.
PLA-Anwendung kann bei optimaler Organisation zur schnelleren Informationsverarbeitung gegenüber einer ROM-Abarbeitung führen.

Die PLA besteht ebenfalls wie ein ROM aus einer Decodierlogik und einer Umcodiermatrix. Sie kann wie ein ROM bei der Herstellung durch Masken oder vom Anwender durch Überstromstärke programmiert werden. Um die Programmierbarkeit durch den Anwender zu gewährleisten, wird bei der PLA die Decodierlogik ebenfalls aus programmierbaren Elementen aufgebaut **(Bild 2.55)**.
Die Matrix 1 der PLA wirkt spaltenweise wie NAND-Gatter. Die Matrix 2 faßt diese NAND-Ausdrücke noch einmal in NAND-Verknüpfungen zusammen, so daß nach dem Theorem von *de Morgan* insgesamt ODER-verknüpfte UND-Ausdrücke (disjunktiv verknüpfte Konjunktionen) entstehen.
Anstelle der im Bild 2.55 dargestellten Ausgangsnegatoren werden oft gesteuerte Inverter (EXCLUSIV-ODER-Glieder) verwendet, so daß dann die Ausgangsvariablen auch negiert auftreten können.

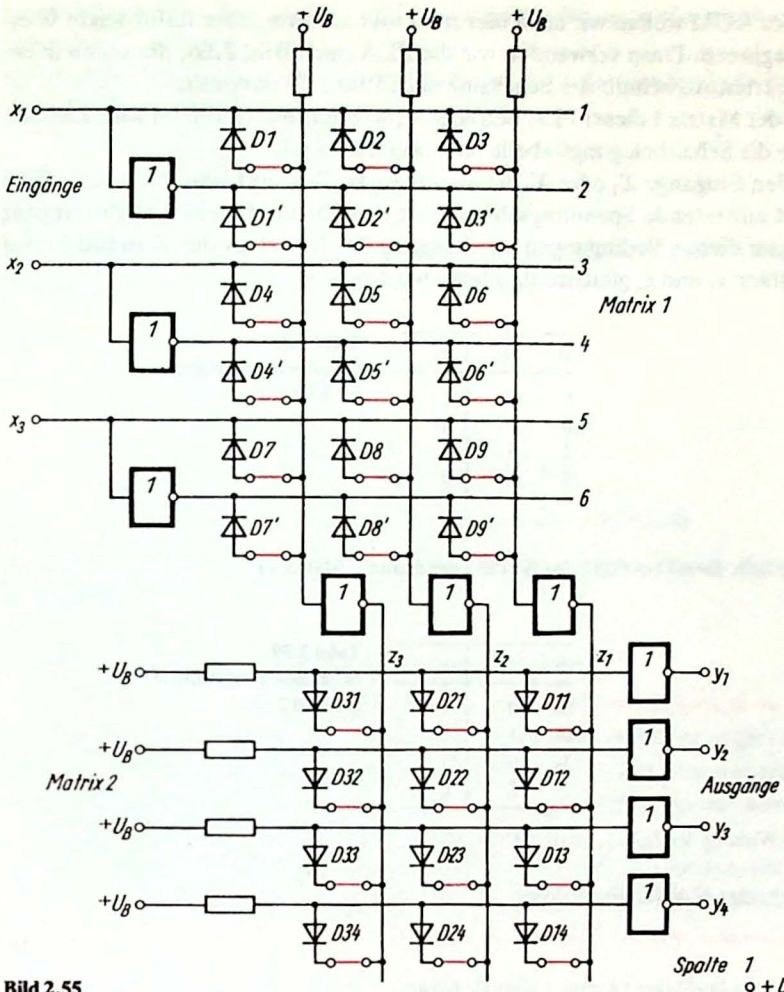


Bild 2.55
Schaltung einer programmierbaren Logikanordnung

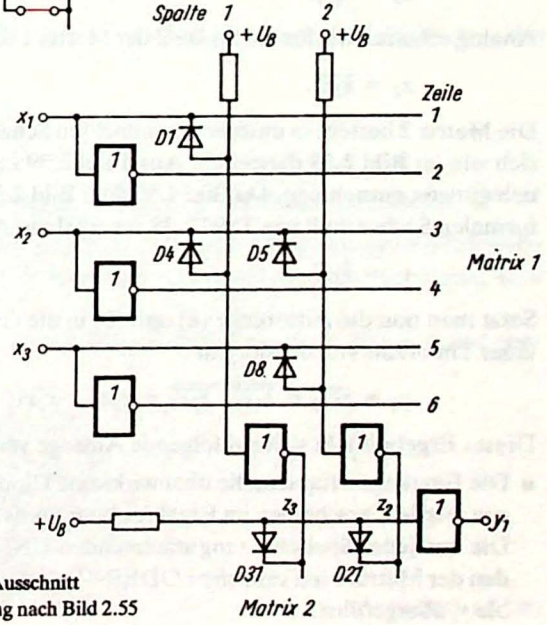


Bild 2.56
In bestimmter Form programmierter Ausschnitt der programmierbaren Logikanordnung nach Bild 2.55

Wie bei der Einführung des ROM wollen wir auch hier mit einer kleinen, aber dafür leicht überschaubaren Anordnung beginnen. Dazu verwenden wir die PLA nach **Bild 2.56**, die einen in bestimmter Form programmierten Ausschnitt der Schaltung nach Bild 2.55 darstellt.
Wenn wir nur die Spalte 1 der Matrix 1 dieser PLA betrachten, so erhalten wir die im **Bild 2.57** dargestellte Schaltung, für die die Schaltbelegungstabelle nach **Tafel 2.38** gilt.
Solange an einem der beiden Eingänge X_1 oder X_2 die Spannung im Zustandsbereich 0 liegt, fließt ein Strom I, und der an R auftretende Spannungsabfall führt den Zustand 0 am Negatoreingang herbei. Damit entsteht unter diesen Bedingungen am Ausgang des Inverters der Zustand 1. Nur wenn beide Eingangsvariablen x_1 und x_2 gleichzeitig 1 sind, wird $z_3 = 0$.

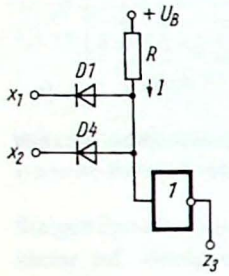


Bild 2.57. Detail aus Bild 2.56 (Wirkung der Spalte 1, Matrix 1)

x_2	x_1	z_3
0	0	1
0	1	1
1	0	1
1	1	0

Tafel 2.38
Schaltbelegungstabelle für Bild 2.57

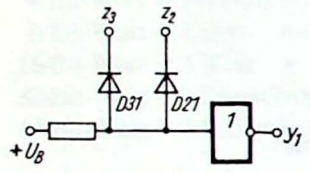


Bild 2.58. Detail aus Bild 2.56 (Wirkung der Zeile 1, Matrix 2)

z_3	z_2	y_1
0	0	1
0	1	1
1	0	1
1	1	0

Tafel 2.39
Schaltbelegungstabelle für Bild 2.58

Dieses Verhalten entspricht der NAND-Funktion

$$z_3 = \overline{x_1 x_2}. \tag{a}$$

Analog erhalten wir für die Spalte 2 der Matrix 1 den Betrag

$$z_2 = \overline{x_2 x_3}. \tag{b}$$

Die Matrix 2 besteht in unserer vereinfachten Schaltung nur aus den Dioden D31 und D21 und läßt sich wie im **Bild 2.58** darstellen. Aus Tafel 2.39 kann man die zueinander gehörenden Variablenbelegungen entnehmen. Da Bild 2.58 dem Bild 2.57 entspricht, muß auch **Tafel 2.39** den gleichen formalen Sachverhalt wie Tafel 2.38 ausdrücken. Auch hier entsteht eine NAND-Funktion

$$y_1 = \overline{z_3 z_2}. \tag{c}$$

Setzt man nun die Ausdrücke (a) und (b) in die Gl. (c) ein, so erhält man unter Berücksichtigung eines Theorems von de Morgan

$$y_1 = \overline{z_3 z_2} = \overline{\overline{x_1 x_2} \cdot \overline{x_2 x_3}} = x_1 x_2 \vee x_2 x_3.$$

Dieses Ergebnis läßt sich auf folgende Aussage verallgemeinern:

- Die Eingangsvariablen, die über wirksame Dioden auf eine gemeinsame Spaltenleitung übertragen werden, erscheinen im Ergebnis konjunktiv (UND) verknüpft.
  Die aus jeder Spaltenleitung stammenden UND-Ausdrücke werden durch die wirksamen Dioden der Matrix 2 in Form einer ODER-Verknüpfung (disjunktiv) in die jeweilige Ausgangsvariable y_i übergeführt.

Sicherlich sind Sie jetzt schon in der Lage, die Entstehung umfangreicher Ergebnisausdrücke zu erkennen.
Machen wir eine Probe: Wie lauten die Ergebnisfunktionen für y_1 und y_2, wenn folgende Dioden im Bild 2.55 wirksam sind:

D1, D2′, D3′, D4′, D6, D9,

D31, D11, D32, D22 ?

Wenn Sie sich dieser Überprüfung unterziehen wollen, so lesen Sie noch nicht weiter.
Das Ergebnis lautet, in Einzelschritte aufgelöst:

$$z_3 = \overline{x_1\bar{x}_2}$$

$$z_2 = \overline{\bar{x}_1} = x_1$$

$$z_1 = \overline{\bar{x}_1 x_2 x_3}$$

$$y_1 = \overline{z_1 z_3} = \overline{\overline{\bar{x}_1 x_2 x_3} \cdot \overline{x_1 \bar{x}_2}} = \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2$$

$$y_2 = \overline{z_2 z_3} = \overline{x_1 \cdot \overline{x_1 \bar{x}_2}} = \bar{x}_1 \vee x_1 \bar{x}_2 = \bar{x}_1 \vee \bar{x}_2 = \overline{x_1 x_2}$$

Zum optimalen Einsatz von PLA werden Minimierungsverfahren angewendet, die wir aber nicht behandeln wollen.

2.1.7.9. Schaltungen zur Addition und Subtraktion

Wir wissen bereits: Zwei Dualzahlen werden addiert, indem man gleichwertige Binärstellen und einen aus der vorhergehenden Stelle evtl. vorhandenen Übertrag nach Tafel 1.12 zusammenfaßt. Die Subtraktion kann in eine Addition übergeführt werden, indem man negative Zahlen und den Subtrahenden im Zweierkomplement darstellt.
Ein ungültiges Ergebnis kann man daran erkennen, daß die Überträge in das höchstwertige Bit hinein und aus dem höchstwertigen Bit hinaus ungleiche Zustände aufweisen.

Zur Realisierung von Addition und Subtraktion mit der gleichen Baugruppe benötigen wir also eine Schaltung, die

- einen Umschalter von Addition auf Subtraktion
- einen Zweierkomplementbilder
- Addierstufen
- einen Auswerter für den Übertrag in das MSB hinein und aus dem MSB hinaus aufweist.

Zunächst wollen wir uns die grundlegende *Addierstufe* für die Addition einer einzigen Binärstelle zweier Dualzahlen ansehen, die in Symbolform im **Bild 2.59** dargestellt ist. Diese Schaltung wird auch als *Volladder* (1 Bit) bezeichnet.
Es gelten folgende Definitionen und Beziehungen zwischen den Variablen des Bildes 2.59 und denjenigen der Tafel 1.12:

$x_1 = a_n$ n-te Binärstelle des 1. Operanden
$x_2 = b_n$ n-te Binärstelle des 2. Operanden
$x_3 = \ddot{u}_{n-1}$ Übertrag von der vorhergehenden Stufe
$y_1 = s_n$ n-te Binärstelle der Summe
$y_2 = \ddot{u}_n$ Übertrag zur nachfolgenden Stufe.

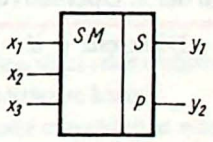


Bild 2.59
Schaltsymbol für Volladder (1 Bit)

Für die Durchführung einer Addition gelten die Beziehungen der Tafel 1.12. Mit ihrer Hilfe können folgende Gleichungen gewonnen werden:

$$s_n = y_1 = x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee x_1x_2x_3. \tag{a}$$

Gl. (a) läßt sich zusammenfassen zu

$$y_1 = \bar{x}_3\,(x_1\bar{x}_2 \vee \bar{x}_1x_2) \vee x_3\,(\bar{x}_1\bar{x}_2 \vee x_1x_2). \tag{b}$$

Für den Übertrag $\ddot{u}_n = y_2$ erhält man

$$\ddot{u}_n = y_2 = x_1x_2\bar{x}_3 \vee x_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee x_1x_2x_3. \tag{c}$$

Gl. (c) läßt sich umformen in

$$y_2 = x_1x_2 \vee x_3\,(x_1\bar{x}_2 \vee \bar{x}_1x_2). \tag{d}$$

Das Signalflußbild einer Volladderstufe für 1 Bit erhält man aus den Gln. (b) und (d) gemäß **Bild 2.60**.

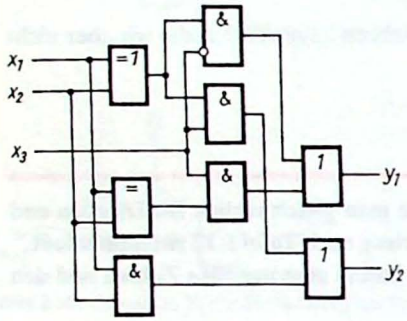


Bild 2.60. Signalflußbild einer Volladderstufe für 1 Bit

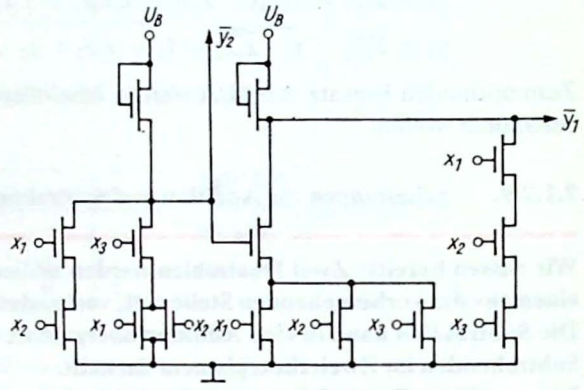


Bild 2.61 Volladderstufe in MOS-Technik

Nach *Möschwitzer/Jorke*[1]) kann man eine Volladderstufe nach **Bild 2.61** realisieren. Dazu muß man die Gln. (a) und (c) wie folgt umschreiben:

$$y_1 = \bar{y}_2\,(x_1 \vee x_2 \vee x_3) \vee x_1x_2x_3$$

$$y_2 = x_1x_2 \vee x_3\,(x_1 \vee x_2).$$

Für die Realisierung eines Volladders (1 Bit) gibt es noch zahlreiche Varianten in den verschiedensten Techniken, die teils nach minimalem Aufwand an Gatterfunktionen, teils nach Gesichtspunkten einer schnellen Addition entworfen wurden. Sollten Sie sich für weitere Ausführungsformen interessieren, ist es das beste, wenn Sie die einschlägigen Patentschriften studieren.

Um zwei mehrstellige Dualzahlen miteinander addieren zu können, benötigt man je Binärstelle einen Volladder. **Bild 2.62** zeigt einen solchen Volladder für 4 Bit.

Wir wollen nun diese Schaltung erweitern auf Subtraktion und Erkennung ungültiger Ergebnisse. Zur Subtraktion muß vom Subtrahenden das Zweierkomplement gebildet werden. Wir wollen festlegen, daß der 2. Operand (b) der Subtrahend sein soll. Dann gilt für die Subtraktion

$$\text{Differenz} = d = a - b.$$

[1]) *Möschwitzer, A.; Jorke, G.:* Mikroelektronische Schaltkreise – Entwurf und Anwendung hochintegrierter Digitalschaltkreise und Mikroprozessoren. 2. Aufl. Berlin: VEB Verlag Technik 1981

Zur Komplementbildung wollen wir die Methode a von S. 28 heranziehen, weil sie sich technisch leicht realisieren läßt. Danach müssen alle Ziffern des Subtrahenden invertiert werden, und anschließend muß in der niedrigsten Stelle (LSB) eine Eins addiert werden.
Die Addition einer Eins im LSB können wir über den freien Übertragseingang der Addierstufe SM0 realisieren.
Die Invertierung aller b_i-Werte kann über Antivalenzschaltungen (EXCLUSIV-ODER), wie auf S. 59 beschrieben, durchgeführt werden. Dabei entscheidet dann die Variable w über die Art der Operation **(Tafel 2.40)**.

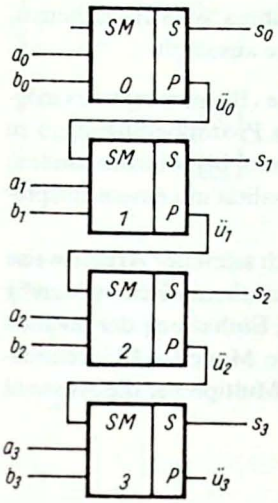


Bild 2.62. Volladder für 4 Bit

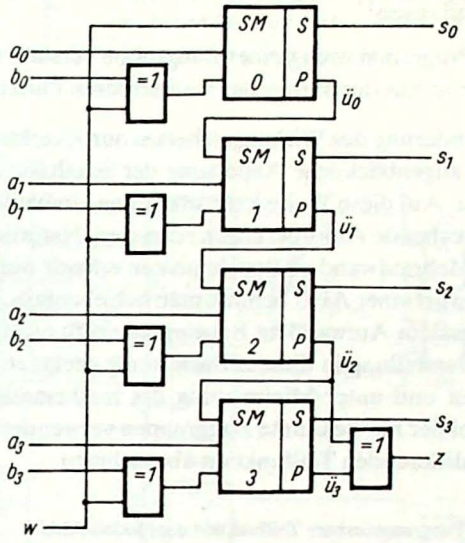


Bild 2.63. Addier/Subtrahier-Schaltung für 4 Bit mit Erkennung ungültiger Ergebnisse

Tafel 2.40
Zustandsdefinitionen für die Addier/Subtrahier-Schaltung nach Bild 2.63

Physikalischer Zustand	Symbolische Zustands-kennzeichnung
Addition	$w = 0$
Subtraktion	$w = 1$
Ergebnis gültig	$z = 0$
Ergebnis ungültig	$z = 1$

Ein EXCLUSIV-ODER-Gatter kann auch die Erkennung ungültiger Ergebnisse vornehmen. Dazu müssen die ins MBS hinein- und hinausgehenden Überträge auf ein solches EXCLUSIV-ODER-Glied gegeben werden. Dieses Glied gibt dann den Zustand 1 am Ausgang ab, wenn der Zustand der Eingangsvariablen ungleich ist. Das vollständige Signalflußbild für eine Addier/Subtrahier-Schaltung für 4 Bit ist im **Bild 2.63** dargestellt. Tafel 2.40 zeigt die Zustandsdefinitionen.

2.1.7.10. Arithmetik-Logik-Einheit (ALU)

Die Arithmetik-Logik-Einheit (arithmetic logic unit = ALU) ist ein Zuordner, der wahlweise arithmetische oder logische Verknüpfungen zwischen zwei Operanden realisieren kann.
Sie ist eine universelle Baugruppe, bei der durch ein Auswahlbitmuster entschieden wird, welche der möglichen Teilfunktionen im gerade laufenden Zeitabschnitt zur Wirkung kommen.

Bereits beim Adder haben wir die Möglichkeit kennengelernt, daß ein und dieselbe Schaltung sowohl addieren als auch subtrahieren kann, wenn man über ein Signal (in unserem Fall w) eine Veränderung der logischen Verknüpfung herbeiführt. Nun werden wir eine umfangreichere Anordnung kennenlernen, bei der unter etlichen möglichen Teilfunktionen eine zur Ausübung ihrer Wirkung ausgewählt werden kann.
Diesen Vorgang, daß eine der möglichen Teilfunktionen einer universellen Baugruppe ausgewählt wird, nennt man ebenfalls *Programmieren der Baugruppe*. So kennen wir nun nach dem Vorgang des Programmierens beim ROM noch eine zweite Auffassung zu diesem Begriff und können verallgemeinernd sagen:

- Unter Programmierung einer Baugruppe versteht man die Festlegung ihres Wirkungsschemas, indem man aus der Menge der realisierbaren Funktionen eine bestimmte auswählt.

Mit der Änderung des Wirkungsschemas durch elektrische Auswahlsignale (Bitmuster) ist es möglich, eine augenblickliche Anpassung der Schaltung an die Umwelt- oder Prozeßbedingungen zu realisieren. Auf diese Weise kann man ohne Umbauten in kurzer Zeit (Nano- bis Mikrosekunden) auf entsprechende Anforderungen reagieren. Natürlich muß diese Universalität mit einem entsprechenden Mehraufwand an Bauelementen erkauft werden.
Beim Entwurf einer ALU bemüht man sich ebenfalls, die Forderungen nach schneller Arbeitsweise und minimalem Aufwand an Bauelementen zu erfüllen. Dieser Weg ist mathematisch aufwendig und zur Darstellung in diesem Buch wenig geeignet. Sie sollen daher bei Einhaltung der leichten Faßlichkeit und unter Minimierung des mathematischen Aufwands eine Modell-ALU kennenlernen, bei der nur bekannte Baugruppen verwendet werden und bei der Multiplexer die Auswahl der zu realisierenden Teilfunktion übernehmen.

Tafel 2.41. Programmierbare Teilfunktion der Modell-ALU

u	v	w	Operation		
0	–	0	$y = x_1 + x_2$	Addition	arithmetischer Teil
0	–	1	$y = x_1 - x_2$	Subtraktion	
1	0	0	$y = x_1 x_2$	UND-Funktion	Logikteil
1	0	1	$y = x_1 \vee x_2$	ODER-Funktion	
1	1	0	$y = x_1 \veebar x_2$	EXCLUSIV-ODER	
1	1	1	$y = \bar{x}_1$	Negation von x_1	

Geben wir zunächst vor, daß unsere ALU 6 Teilfunktionen (wie in **Tafel 2.41** dargestellt) realisieren kann. Es handelt sich um 2 arithmetische und 4 logische Operationen, die durch das Bitmuster der drei Auswahlvariablen u, v, w ausgewählt werden sollen. Bei $u = 0$ sollen die arithmetischen, bei $u = 1$ die logischen Verknüpfungen wirksam sein. Die Variation von v und w entscheidet dann innerhalb der beiden Operationsteilgebiete über die spezielle Operationsart. So werden z. B. bei $u = w = 0$ die Addition, bei $u = 1$ und $v = w = 0$ das logische UND ausgewählt usw.
Befassen wir uns nun mit dem Entwurf des Logikteils, den wir nach **Bild 2.64** realisieren wollen. *Vergleichen Sie dieses Bild einmal mit dem auf Seite 61 im Bild 2.28 dargestellten 4:1-Multiplexer.* Sie werden erkennen, daß beide Signalflußbilder nach dem gleichen Grundkonzept aufgebaut sind. Während im Bild 2.28 nur jeweils eine Variable vor jedem Übertragungseingang steht, sind es im Bild 2.64 die Ergebnisse der logischen Verknüpfung der Variablen x_1, x_2. Dabei sind die zur Realisierung der ODER-und EXOR-Funktion benötigten Verknüpfungsglieder vor die zugehörigen Übertragungseingänge geschaltet. Die UND-Funktion kann durch Einbeziehung in den sowieso benötigten UND-Glied-Torbaustein realisiert werden. Die Negation von x_1 ist nur durch den Negationspunkt am zugehörigen UND-Tor dargestellt.

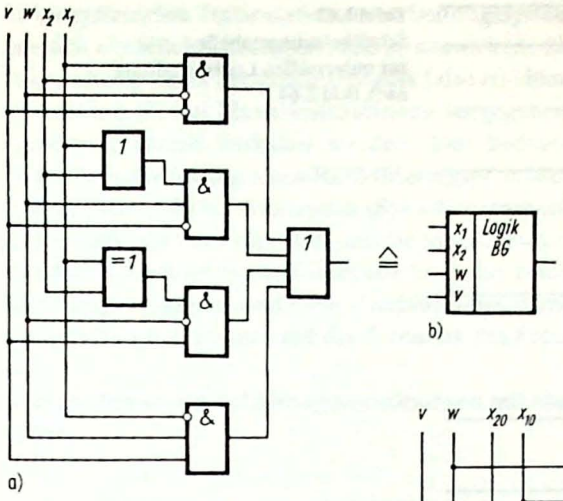


Bild 2.64
Universeller Logikbaustein zur Realisierung der Schaltbelegungstabelle nach Tafel 2.42
a) Signalflußbild; b) Logiksymbol

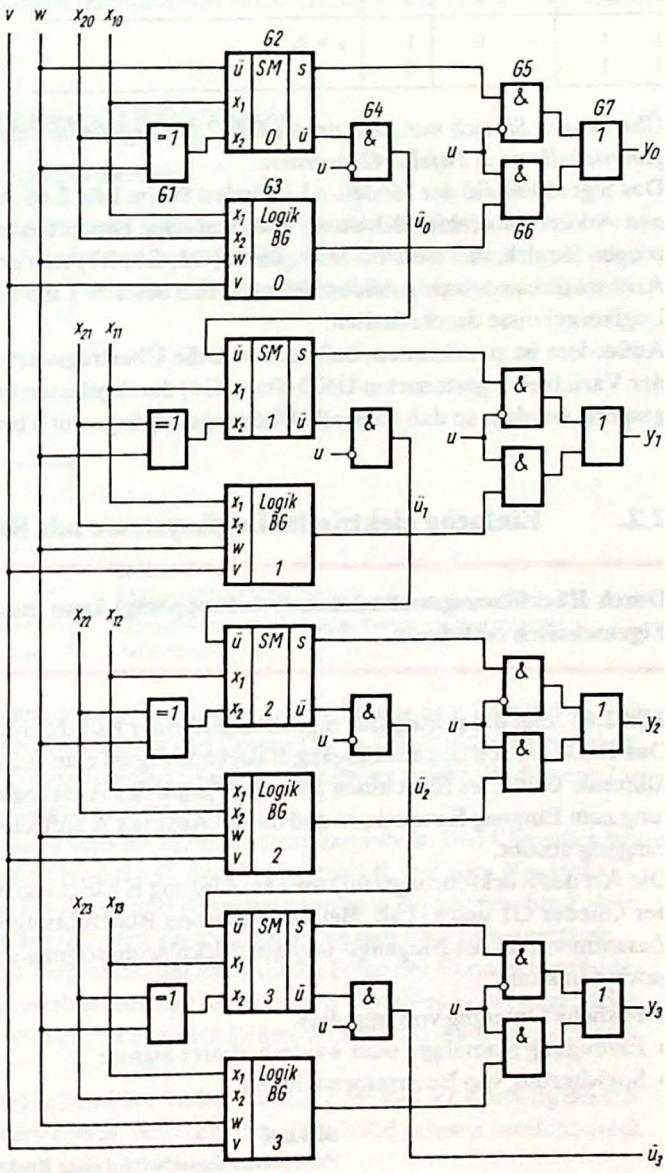


Bild 2.65
4-Bit-Modell-ALU nach Tafel 2.41

v	w	x_2	x_1	y	Funktion
0	0	0	0	0	$y = x_1 x_2$ UND
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	0	$y = x_1 \vee x_2$ ODER
0	1	0	1	1	
0	1	1	0	1	
0	1	1	1	1	
1	0	0	0	0	$y = x_1 \veebar x_2$ EXOR
1	0	0	1	1	
1	0	1	0	1	
1	0	1	1	0	
1	1	–	0	1	$y = \bar{x}_1$
1	1	–	1	0	

Tafel 2.42
Schaltbelegungstabelle zur universellen Logikschaltung nach Bild 2.64

Überzeugen Sie sich nun, daß der im Bild 2.64 dargestellte universelle Logikbaustein die Schaltbelegungstabelle nach ***Tafel 2.42*** *realisiert.*

Das Signalflußbild der Modell-ALU finden Sie im **Bild 2.65**. Es besteht aus der bereits besprochenen Addier/Subtrahier-Schaltung und dem eben entworfenen universellen Logikbaustein. Überzeugen Sie sich, daß die 2 : 1-Multiplexer (G5, G6, G7) vor den y-Ausgängen bei $u = 0$ das von den Arithmetikbausteinen gebildete Resultat und bei $u = 1$ die von den Logikbaugruppen gebildeten Logikergebnisse durchschalten.

Außerdem ist zu erkennen, daß bei $u = 0$ die Übertragswerte über die nachgeschalteten und von der Variablen u gesteuerten UND-Tore (G4) durchgelassen und bei $u = 1$ (Logikteil in Funktion) gesperrt werden, so daß dann alle Übertragsausgänge mit 0 belegt sind.

2.2. Einfache elektrische Logiksysteme mit Rückführung

Durch Rückführungsmaßnahmen (Rückkopplung) kann man Logikschaltungen mit besonderen Eigenschaften realisieren.

Bild 2.66 zeigt das prinzipielle Signalflußbild einer Rückführungsschaltung.

Das Glied G1 soll das am Eingang E liegende Signal zum Ausgang A übertragen. G2 ist das rückführende Glied. Es führt einen gewissen Anteil des Ausgangssignals nach entsprechender Bewertung zum Eingang E zurück, so daß die am Ausgang A auftretende Wirkung auch einen Einfluß am Eingang ausübt.

Die Art des Rückführungseinflusses am Eingang E hängt von der Struktur und der Kompliziertheit der Glieder G1 und G2 ab. Bei den einfachen Rückführungsmaßnahmen wirkt sich das zeitliche Zusammenspiel des Eingangs- und des Rückführungssignals so aus, daß man folgende Funktionen gewinnen kann:

- zeitliche Dehnung von Impulsen
- Erzeugung einmaliger oder wiederholbarer Signale
- Speicherung von Binärinformationen.

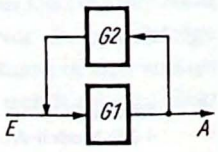


Bild 2.66
Prinzipielles Signalflußbild einer Rückführungsschaltung

Bei komplizierten Informationsverarbeitungssystemen, wie u.a. bei einem Mehrrechnersystem, kann sich ein Rückführungsprozeß so auswirken, daß bestimmte Befehlsanweisungen zur völligen Umgestaltung eines Programmablaufs (also zu einer völlig neuen Wirkungsweise) führen können. So können z.B. bei Havariesituationen vorgegebene Rettungsprogramme anstelle der normalen Arbeitsprogramme wirksam werden. Das bedeutet, daß sich höherentwickelte Informationssysteme in Auswirkung eines Rückführungsprozesses an gegebene Situationen anpassen können.
Auch der sprachliche Austausch (Sprachkommunikation) zwischen Menschen ist ein Rückkopplungsprozeß, der zur Folge hat, daß die Menschen auf die übermittelten Informationen reagieren.
Bei allen Rückführungsmaßnahmen muß das zeitliche Zusammenspiel zwischen dem ursprünglichen Eingangssignal und dem Rückführungsanteil exakt geplant und korrekt realisiert werden. Man spricht auch davon, daß die *Dynamik* des Prozeßablaufs in den Mittelpunkt der Betrachtung rückt.
Im folgenden sollen Schaltungsanordnungen mit einfachen Rückführungsmaßnahmen besprochen werden.

2.2.1. Elementare Logikschaltungen mit Rückführung

2.2.1.1. Monostabiler Multivibrator (Impulsdehner)

Als Beispiel wollen wir eine Schaltung kennenlernen, die einen Impuls der Dauer δ_{1E} zu einem *Rechteckimpuls* der längeren Dauer δ_{1A} *umformen* kann. Im **Bild 2.67** sind das Schaltungssymbol und ein Zeitdiagramm für den prinzipiellen Verlauf von Eingangs- und Ausgangsspannung dargestellt.

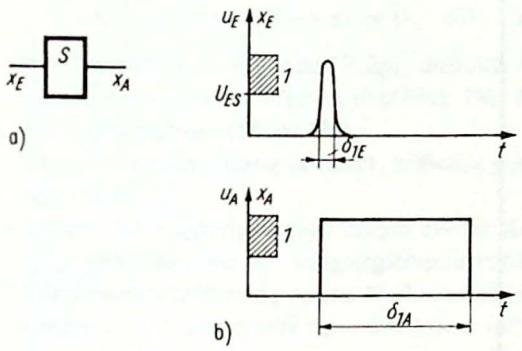


Bild 2.67
Monostabiler Multivibrator
a) Schaltsymbol; b) prinzipieller Verlauf von Eingangs- und Ausgangsspannung

Grobbeschreibung: Hat die Eingangsspannung U_E den Wert U_{ES} erreicht, was dem Zustand $x_E = 1$ entspricht, so folgt auch die Spannung am Ausgang auf den Zustand $x_A = 1$ und verbleibt in diesem Zustand so lange, wie es die innere *Verzögerungseinrichtung* der Schaltung zuläßt. Bedingung für die richtige Funktionsweise ist, daß δ_{1E} kleiner sein muß als δ_{1A}.
Im **Bild 2.68** ist das Signalflußbild eines monostabilen Multivibrators dargestellt. Das Kernstück besteht aus dem NAND-Glied G2, dem Kondensator C, dem Widerstand R und dem Negator G3. Die Negatoren G1 und G4 dienen dazu, Eingangs- und Ausgangsimpulse in die gewünschte Polaritätslage zu bringen. Bild 2.68b zeigt die zugehörigen Zeitdiagramme einiger Spannungsverläufe.
Im **Bild 2.69** ist der Programmablaufplan dargestellt, der die zeitliche Folge der Variablenveränderungen erkennen läßt. In diesem Plan ist zweimal anstelle der logischen Variablen x_4 der zugehörige Spannungsverlauf $u_4(t)$ aufgenommen worden, da in diesen Fällen die zeitliche Änderung der analogen Spannung von Bedeutung ist.
Diskutieren wir nun die einzelnen Phasen anhand der Bilder 2.68 und 2.69. Zur Verkürzung der Erklärung wollen wir ein Pfeilzeichen → verwenden. Anstelle dieses Pfeils soll gelesen werden: „und damit folgt".

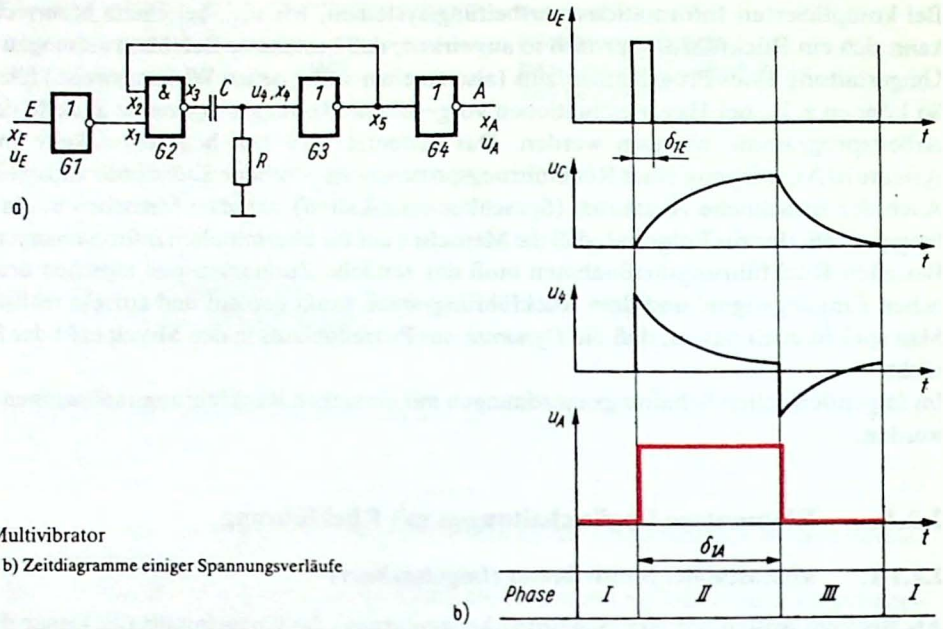


Bild 2.68
Monostabiler Multivibrator
a) Signalflußbild; b) Zeitdiagramme einiger Spannungsverläufe

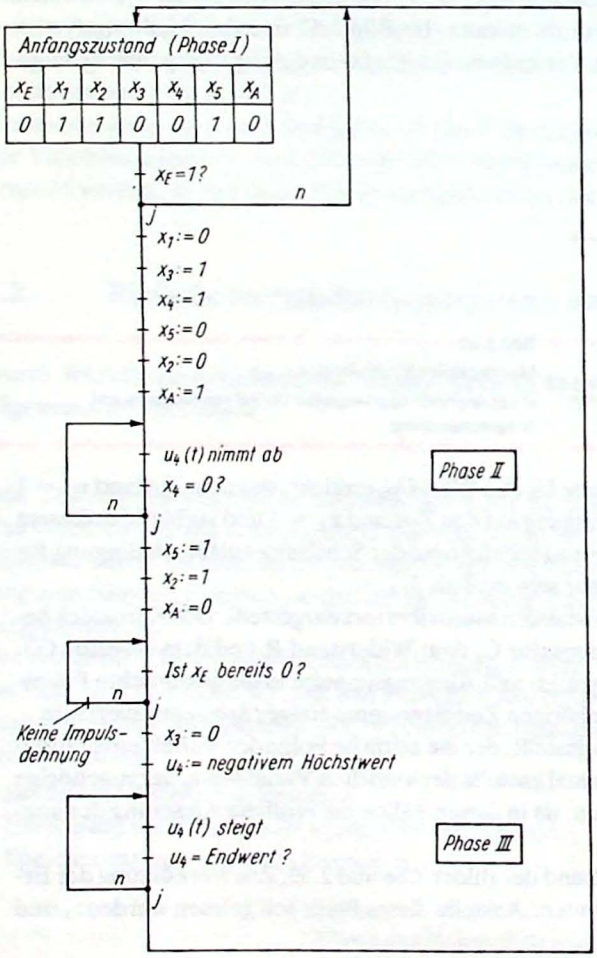


Bild 2.69
Monostabiler Multivibrator, Programmablaufplan

Wir beginnen mit der Phase 1, der Ruhelage. In dieser Phase ist $x_3 = 0$. Es fließt demnach kein Strom durch R, so daß auch $u_4 = 0$ ist. Es besteht in diesem Zustand folgende Verteilung der Variablen:

$$x_E = 0 \rightarrow x_1 = 1$$

$$x_4 = 0 \rightarrow x_5 = x_2 = 1 \rightarrow x_A = 0.$$

Aus $x_1 = x_2 = 1$ folgt, daß der Zustand $x_3 = 0$ stabil bleibt (NAND-Glied).
Kommt nun ein Eingangsimpuls $x_E := 1$, so folgt damit

$$x_1 := 0 \rightarrow x_3 = 1.$$

Da der Kondensator bei sprunghafter Spannungsänderung zunächst wie ein Kurzschluß wirkt, wird der volle Spannungssprung auf den Widerstand R übertragen. Damit folgt die Phase II:

$$u_4 \approx U_B \quad \text{(Batteriespannung)}$$

$$x_4 = 1 \rightarrow x_5 = x_2 = 0 \rightarrow x_A = 1.$$

Der Kondensator lädt sich dann während der Phase II gemäß der Zeitkonstanten $\tau = CR$ auf, und in dem Maße, wie die Spannung am Kondensator wächst, nimmt die Spannung an R ab. Die Spannung u_4 geht also allmählich gegen Null. Der Normalfall sieht nun so aus, daß die Eingangsvariable x_E wieder den Zustand $x_E = 0$ annimmt, ehe x_4 seinen Nullzustand erreicht.
Mit $x_E = 0$ folgt $x_1 = 1$. Trotz dieser Änderung bleibt dann x_3 unverändert ($x_3 = 1$), weil wegen $x_4 = 1$ und damit $x_5 = x_2 = 0$ immer noch eine Variable am Eingang des NAND-Gliedes G2 Null bleibt.
Tritt nun x_4 in den Zustand $x_4 = 0$, so folgt

$$x_5 = x_2 = 1 \rightarrow x_A = 0, \quad \text{d.h.} \quad x_2 = x_1 = 1 \rightarrow x_3 = 0.$$

Der Zustand $x_3 = 0$ hat zur Folge, daß sich der Kondensator C über R entlädt und u_4 damit einen Sprung zu negativen Werten vollführt. Der Entladestrom fließt in der umgekehrten Richtung wie der Aufladestrom (Phase III).
Ist der Entladevorgang beendet, befindet siche der monostabile Multivibrator wieder in der Ruhelage (Phase I).
Dauert der Eingangsimpuls länger als die Kondensatoraufladung, so wird der Zustand $x_3 = 1$ so lange gehalten, wie der Eingangsimpuls vorhanden ist. Unabhängig davon geht aber x_4 wegen der Kondensatoraufladung gegen Null, so daß nur die Aufladezeit den Zeitpunkt bestimmt, bei dem wieder $x_5 = 1$ und damit $x_A = 0$ werden. Für diesen Fall gibt es folglich keine Impulsdehnung.

2.2.1.2. Astabiler Multivibrator (Pulsgenerator)

Ein astabiler Multivibrator ist ein Glied, das einen *Rechteckimpuls* (Rechteckwelle) *erzeugt*, **Bild 2.70** zeigt das Schaltsymbol und das Zeitdiagramm. Ein solches Glied kann zur Erzeugung einer periodischen Taktimpulsfolge verwendet werden.

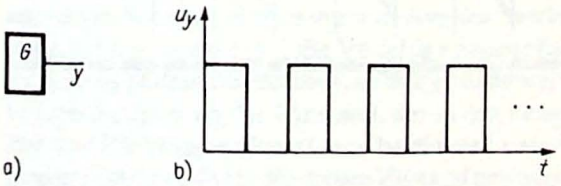


Bild 2.70
Astabiler Multivibrator
a) Schaltsymbol; b) Zeitdiagramm der Ausgangsspannung

Im **Bild 2.71a** ist eine mögliche Schaltungsausführung nach Art eines Signalflußbilds dargestellt. Zwei NAND-Glieder, die hier als Negatoren wirksam sind, übertragen ihre Ausgangsspannungen über CR-Glieder jeweils auf den Eingang des anderen Gliedes. Durch die kapazitive Kopplung bedingt, kann diese Schaltung nur *kurzzeitig stabile* Zustände einnehmen und muß von einem zum

anderen Zustand umkippen. Die zwischenstabilen Zustände sind dadurch gekennzeichnet, daß jeweils eine Stufe am Ausgang den Nullzustand aufweist, während sich die andere dann im Zustand 1 befindet.

Zur Erklärung der Wirkungsweise wollen wir annehmen, daß zu Beginn unserer Betrachtung x_2 einen Sprung von 0 nach 1 macht. Dieser Sprung wird direkt auf x_3 übertragen. Damit geht x_4 in den Zustand 0 (Voraussetzung: $s = 1$), und durch die Kondensatorkopplung springt auch x_1 in den Zustand 0, wodurch $x_2 = 1$ folgt. $x_2 = 1$ entspricht aber unserer Ausgangsannahme. In diesem Zustand verbleibt die Schaltung so lange, bis die Umladestromstärke von C1 so gering geworden ist, daß x_3 den Zustand 0 einnimmt. Dann vollzieht sich folgende Kettenreaktion:

$$\rightarrow x_3 := 0 \rightarrow x_4 := 1 \rightarrow x_1 := 1 \rightarrow x_2 := 0 \rightarrow$$

Damit ist dann der nächste zwischenstabile Zustand erreicht, der dann beendet wird, wenn x_1 wieder den Zustand $x_1 = 0$ annimmt.

An den Ausgängen A1 und A2 treten Spannungsschwankungen auf, die im **Bild 2.71 b** als Zeitverläufe dargestellt sind.

Bringt man im Bild 2.71 a den Eingang S in den Zustand $s = 0$, so stoppt der Generator. Er schwingt nur bei $s = 1$.

Nach *[Kühn/Schmied]*[1]) muß man, wie im **Bild 2.71 c** gezeigt, noch die Glieder G3 und G4 einschalten, falls man die Widerstände unterhalb eines Grenzwiderstandswerts (bei TTL-Schaltkreisen ist $R_{grenz} = 1{,}7\ k\Omega$) wählt, damit das sichere Anschwingen des Generators gewährleistet wird.

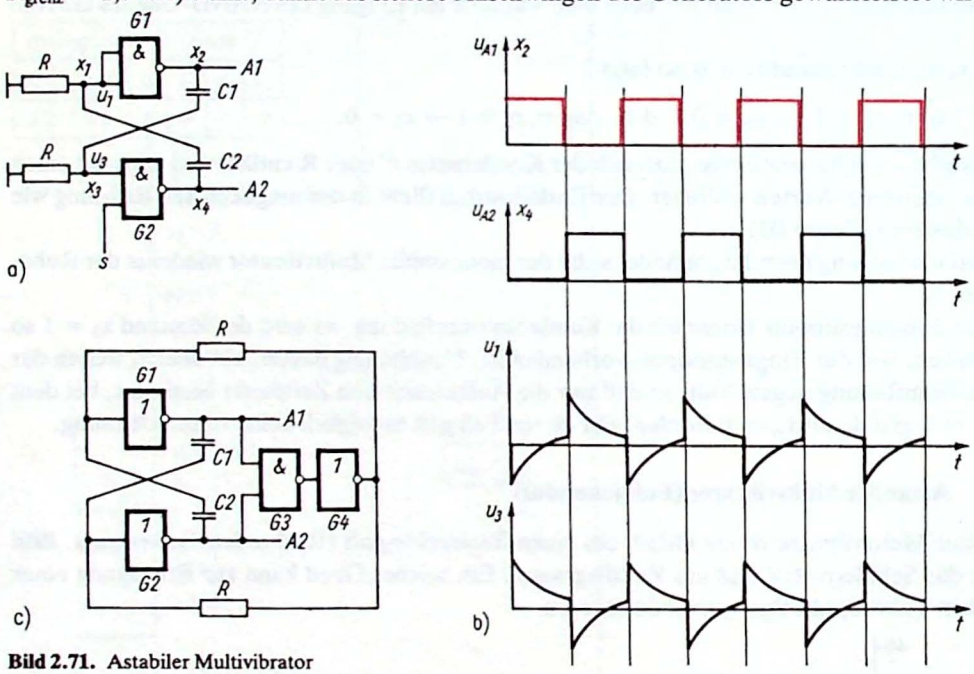


Bild 2.71. Astabiler Multivibrator

a) allgemeines Signalflußbild; b) Zeitdiagramme einiger Spannungsverläufe; c) spezielles Signalflußbild für TTL-Schaltkreise, um bei Widerständen unter 1,7 kΩ das sichere Anschwingen zu gewährleisten

2.2.1.3. RS-Trigger aus NAND-Gliedern

Läßt man beim astabilen Multivibrator die Widerstände weg und ersetzt die Kondensatoren durch Kurzschlüsse, so erhält man eine Schaltung, die in *zwei stabilen* Zuständen verharren kann. Diese Schaltung wird RS-Trigger oder auch RS-Flipflop genannt **(Bild 2.72)**. S kommt von set ≙ setzen und R von reset ≙ rücksetzen.

Wie aus **Tafel 2.43** zu erkennen ist, unterscheiden sich die beiden stabilen Lagen dadurch, daß in der Setzlage $a_1 = 1$ und $a_2 = 0$ und in der Rücksetzlage $a_1 = 0$ und $a_2 = 1$ sind. Die in der s- bzw. r-Spalte angegebene Kombination 1,0,1 soll ausdrücken, daß die jeweilige stabile Lage durch eine Variablenbelegungsfolge Zustand 1, gefolgt von Zustand 0, gefolgt von Zustand 1 erreicht worden ist. In beiden stabilen Lagen weisen s und r nach dem Umschalten in die jeweilige Lage folglich beide den Zustand 1 auf.

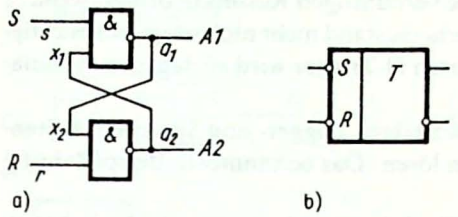


Bild 2.72
RS-Trigger
a) Signalflußbild; b) Schaltsymbol

Tafel 2.43. Zustandstabelle für den RS-Trigger nach Bild 2.72

	s	r	x_1	x_2	a_1	a_2	
Setzlage	1,0,1	1	0	1	1	0	
Rücksetzlage	1	1,0,1	1	0	0	1	

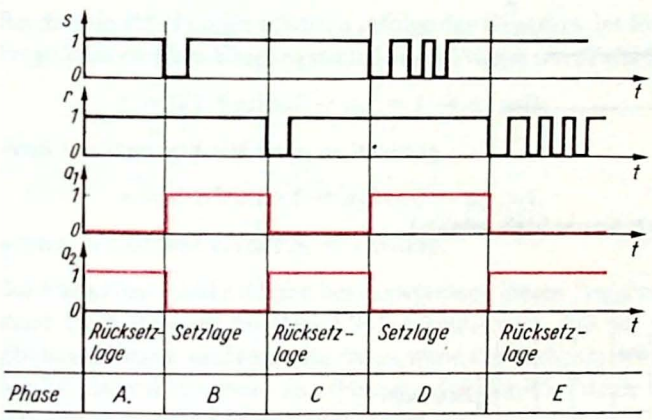


Bild 2.73
RS-Trigger, Zeitdiagramm

Bild 2.73 zeigt diesen Zusammenhang noch einmal in einem Zeitdiagramm. In der Phase A besteht die Rücksetzlage ($a_1 = 0, a_2 = 1$). Es sind $s = 1$ und $r = 1$. Wird nun s gegen 0 geführt, so muß im Bild 2.72a am Ausgang A1 die Variable $a_1 = 1$ werden.

$$a_1 = 1 \rightarrow x_2 = 1 \rightarrow a_2 = 0 \rightarrow x_1 = 0$$

Nimmt nun s wieder den Zustand $s = 1$ an, so bleibt wegen $x_1 = 0$ der Zustand $a_1 = 1$ erhalten. Damit ist die Setzlage eingenommen. Aus der Setzlage gelangt der RS-Trigger in die Rücksetzlage, wenn bei $s = 1$ und $r = 1$ die Variable r gegen Null geht. Der Vorgang des Rücksetzens erfolgt dabei analog zu dem des Setzens, so daß er nicht weiter erklärt zu werden braucht.
Interessant ist noch der Umstand, der in den Phasen D und E (Bild 2.73) demonstriert worden ist: Hat der RS-Trigger einmal eine bestimmte Lage eingenommen, so kann er durch diejenige Eingangsvariable s oder r, die diesen Zustand herbeigeführt hat, *nicht* mehr umgeschaltet werden. Der zugehörige Ausgang bleibt im Zustand 1, weil die vom anderen Ausgang rückgekoppelte Variable 0 ist und der Zustand 0 beim NAND-Glied *durchgreift* (s. S. 54). So sieht man z. B. in der Phase D, daß nach Einnehmen der Setzlage Schwankungen der Variablen s nichts mehr ändern. Ebenso ist es in der Phase E bei Schwankungen der Variablen r.

Eine Lage bleibt also so lange erhalten, bis die diesen Zustand nicht verursachende Variable einen 1,0,1-Verlauf durchmacht. Dem Zustand eines RS-Triggers kann man also entnehmen, welche Variable (s oder r) die eingenommene Lage verursacht hat. Damit liegt ein *Speicherprozeß* vor!
Es gibt noch einen Zustand, der dadurch gekennzeichnet ist, daß beide Eingangsvariablen s und r gleichzeitig Null sind, wonach folgt, daß beide Ausgänge den Zustand 1 aufweisen. Dann entscheidet diejenige Eingangsvariable die Speicherlage des RS-Triggers, die zuletzt in den Zustand 1 gelangt. Dieser Zustand hat den Nachteil, daß er dann keine vernünftigen Resultate bringt, wenn er rein zufällig auftritt, weil der danach eingenommene Speicherzustand nicht mehr dem beabsichtigten Ergebnis entsprechen muß. Beim taktflankengesteuerten D-Trigger wird er dagegen systematisch genutzt.
Der RS-Trigger ist ein beliebtes Grundelement für kompliziertere Trigger- und Speicherschaltungen. Aber auch mit ihm allein kann man schon Aufgaben lösen. Das bekannteste Beispiel dürfte der Aufbau eines *prellfreien* Schalters sein.
Schalter prellen beim Einschalten. Anstelle des gewünschten einmaligen Stromsprungs kommt es zu mehreren Ein- und Ausschaltungen, bis der endgültige Einschaltzustand erreicht ist **(Bild 2.74)**.

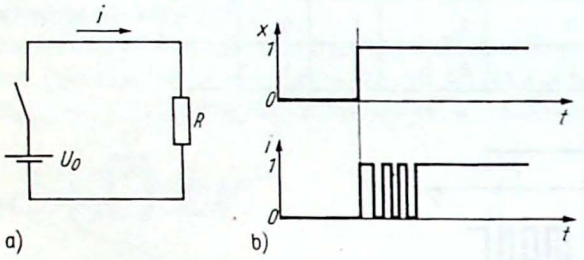
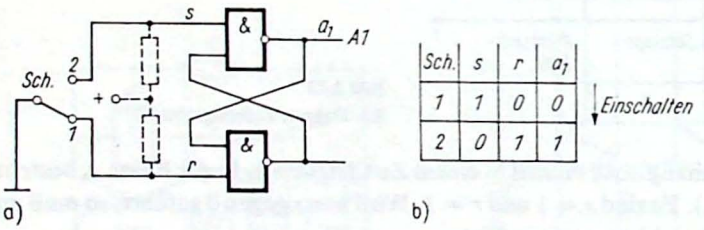


Bild 2.74
Prellwirkung eines Schalterkontakts
a) Schaltung; b) Zeitdiagramme (die Variable x gibt die Stellung des Schalterhebels an)



Sch.	s	r	a_1	
1	1	0	0	Einschalten
2	0	1	1	

b)

Bild 2.75. Beseitigung des Prellens durch RS-Trigger
a) Schaltung; b) Schaltbelegungstabelle

Mit Hilfe eines RS-Triggers kann das Prellen vollständig beseitigt werden **(Bild 2.75)**. Man benötigt dazu einen Umschalter. Die gestrichelt eingezeichneten Widerstände brauchen bei TTL-Schaltungen nicht mit eingeschaltet zu werden. Sie sollen nur verdeutlichen, wie die Potentialumschaltung vor sich geht.
Zuerst wird der Umschalter in die Stellung 1 gebracht. Damit nimmt der RS-Trigger die Rücksetzlage ($a_1 = 0$) ein. Schaltet man danach auf Schalterstellung 2, so geht der Trigger in die Setzlage ($a_1 = 1$). Sollte der Schalter am Kontakt 2 prellen, bleibt mit Wirkung der ersten Kontaktgabe $a_1 = 1$. Damit ist am Ausgang A1 ein prellfreier 0/1-Sprung vollzogen worden.
Auf diese Weise kann man fehlerfreie Zustandseingaben mit der Hand (Handeingaben) an digitale Informationsverarbeitungsgeräte realisieren.

2.2.1.4. RS-Trigger aus NAND-Gliedern mit vorgeschalteten Negatoren

Schaltet man vor beide Eingänge des RS-Triggers einen Negator, so wird damit der Übergang zum *getakteten* RS-Trigger vorbereitet. Gleichzeitig kann man mit dieser Schaltung die Wirkungsweise des RS-Triggers noch besser verstehen **(Bild 2.76)**.

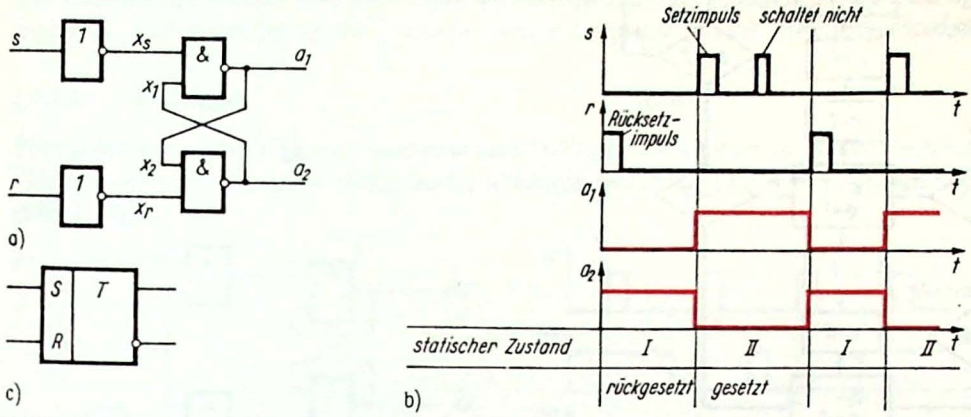


Bild 2.76. RS-Trigger mit Negation der Eingangsvariablen
a) Signalflußbild; b) Zeitdiagramme; c) Schaltsymbol

Bei diesem RS-Trigger schalten infolge der Negation der Eingangsvariablen die 0/1-Flanken der in Frage kommenden Eingangssignale den Trigger um **(Tafel 2.44)**:

$$s = 0/1\text{-Verlauf} \rightarrow a_1 := 1 \rightarrow a_2 := 0,$$

wenn $r = 0$ ist und vorher $a_1 = 0$ vorlag.

$$r = 0/1\text{-Verlauf} \rightarrow a_1 := 0 \rightarrow a_2 := 1,$$

wenn $s = 0$ ist und vorher $a_1 = 1$ vorlag.

Die Variablen s und r dürfen bei Anwendung dieses Triggers als Speicher nicht gleichzeitig den Zustand 1 einnehmen! Im Bild 2.76b erkennt man, daß nur positive Eingangsimpulse den Trigger schalten können, und zwar nur dann, wenn der durch sie zu erreichende Zustand noch nicht vorhanden ist. Hieran ist erneut zu erkennen, daß der RS-Trigger den Zustand speichert, den der zuletzt wirksam gewordene Impuls verursacht hat.

Tafel 2.44. Zustandstabelle für den RS-Trigger mit negierten Eingangsvariablen nach Bild 2.76

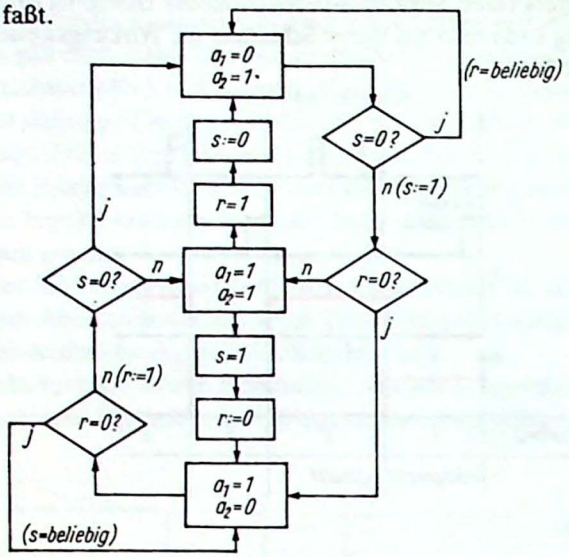
	s	r	x_s	x_r	a_1	a_2
Setzlage	0,1,0	0	1,0,1	1	1	0
Rücksetzlage	0	0,1,0	1	1,0,1	0	1

Die Speicherfähigkeit des RS-Triggers kann leider auf drei Arten gestört werden:

- bei Abschalten der Batteriespannung
- bei Auftreten eines Störimpulses, der den Speicherinhalt verändert
- wenn gleichzeitig an beiden Eingängen der den Speichervorgang gefährdende Zustand liegt.

Bei Abschalten der Batteriespannung werden alle Variablen zu Null, und damit geht das *Gedächtnis* verloren. Störimpulse können falsche Speicherinhalte erzeugen oder wahre Speicherinformationen löschen. Liegt beim RS-Trigger mit negierten Eingangsvariablen an beiden Eingängen der Zustand 1, so passiert das, was beim RS-Trigger unter dem Aspekt besprochen worden ist, daß beide Eingangsvariablen den Nullzustand aufweisen.

In einem Zustandsdiagramm **(Bild 2.77)**, das wie ein Programmablaufplan auszuwerten ist, nur daß es in diesem Fall keinen vorgegebenen Anfang gibt, sind noch einmal alle möglichen Zustände erfaßt.



Zeichenerklärung:

= *Kennzeichnung der erreichten Zustände der Ausgangsvariablen*

= *Operation bei :=*
Zustandskennzeichnung bei =

= *Frage nach dem Zustand einer Eingangsvariablen*

Bild 2.77
RS-Trigger mit Negation der Eingangsvariablen, ausführliches Zustandsdiagramm

2.2.1.5. Getakteter RS-Trigger

Aus dem RS-Trigger mit negierten Eingangsvariablen wird ein *getakteter* RS-Trigger, wenn anstelle der Negatoren NAND-Glieder verwendet werden. Über einen Takteingang C (C von clock ≙ Zeitgeber) werden die beiden Eingangs-NAND-Glieder als Tore betrieben **(Bild 2.78)**.

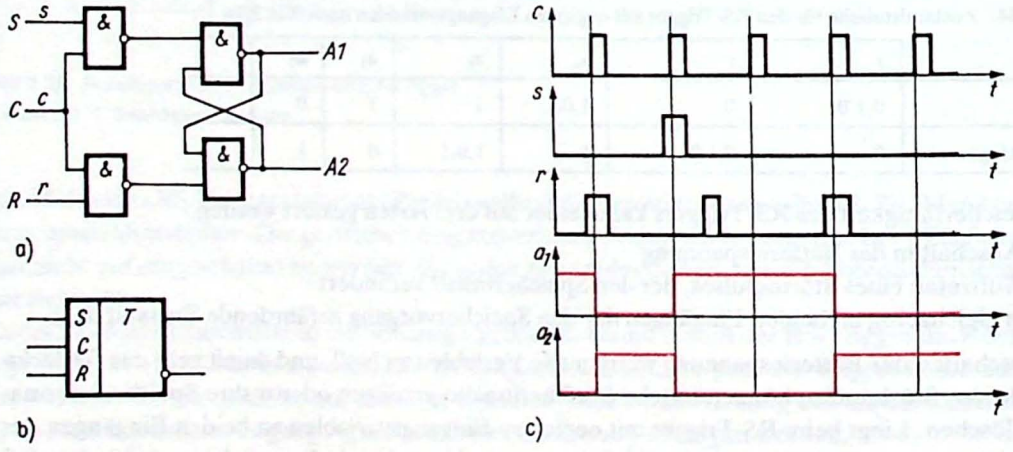


Bild 2.78. Getakteter RS-Trigger
a) Signalflußbild; b) Schaltsymbol; c) Zeitdiagramme

Mit dieser Schaltungsmaßnahme können die an S oder R anliegenden Stellimpulse erst dann wirksam werden, wenn auch am Eingang C der positive Taktimpuls vorhanden ist. Durch diese Maßnahme werden alle Störimpulse, die über die Zuleitungen an S und R gelangen, unwirksam gemacht, sofern sie nicht zeitlich mit dem Taktimpuls zusammenfallen. Da man die Taktimpulse mit sehr kleiner Impulsdauer erzeugen kann, wird dadurch die Störanfälligkeit der Schaltung stark herabgesetzt. Außerdem bringt der Takt eine straffe Zeitplanung in die Ablauforganisation hinein.

2.2.1.6. D-Trigger

Vom getakteten RS-Trigger gelangt man zum D-Trigger, wenn man die beiden Eingangsvariablen r und s über einen Negator voneinander abhängig macht, also die Verknüpfung realisiert $r = \bar{s}$ **(Bild 2.79)**.

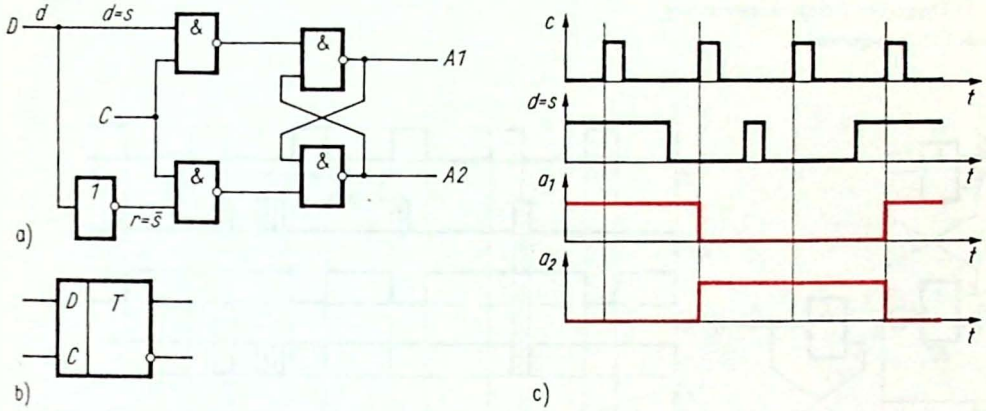


Bild 2.79. D-Trigger
a) Signalflußbild; b) Schaltsymbol; c) Zeitdiagramme

Mit $r = \bar{s}$ werden die beim RS-Trigger möglichen Zustände $s = r = 1$ und $s = r = 0$ vermieden, so daß diese, mitunter unerwünschten Zustände nicht eingenommen werden können.
Auch der D-Trigger kann nur umgeschaltet werden, wenn ein positiver Taktimpuls erscheint. Bei $d = 1$ wird mit dem Taktimpuls die Ausgangsvariable $a_1 = 1$ gesetzt. Dieser Zustand hält bis zum nächsten Taktimpuls an. Hat sich inzwischen $d = 1$ nicht verändert, so bleibt $a_1 = 1$. Ist aber in der Zwischenzeit d zu Null geworden, so wird beim Eintreffen des nächsten Taktimpulses a_1 von 1 nach 0 geschaltet, also der Trigger zurückgesetzt. Jeder Taktimpuls stellt also, falls erforderlich, den Trigger neu ein (Bild 2.79c).
Diese Form des D-Triggers hat noch den Nachteil, daß der Zustand am Eingang D während der Taktimpulsdauer konstant bleiben muß, weil sonst mehrmals umgeschaltet werden könnte und dann erst die letzte konstante Phase der Variablen d entscheidet, ob der Trigger gesetzt oder rückgesetzt wird.

2.2.1.7. D-Trigger mit Taktflankensteuerung

Der D-Trigger mit Taktflankensteuerung **(Bild 2.80)** übernimmt den Zustand des D-Eingangs beim 0/1-Übergang des Taktimpulses und speichert ihn in der Ausgangsvariablen a_1 für die Dauer bis zum Eintreffen des 0/1-Übergangs des nächsten Taktimpulses. Ändert sich der Zustand des D-Eingangs während der Taktimpulsdauer, so behält die Ausgangsvariable trotzdem ihren während der Taktflanke eingenommenen Zustand. Der beim gewöhnlichen D-Trigger dargestellte Nachteil entfällt also.
Dieser Trigger eignet sich besonders für getaktete Systeme, um asynchron eintreffende Impulse mit der 0/1-Flanke des Taktimpulses zu synchronisieren.

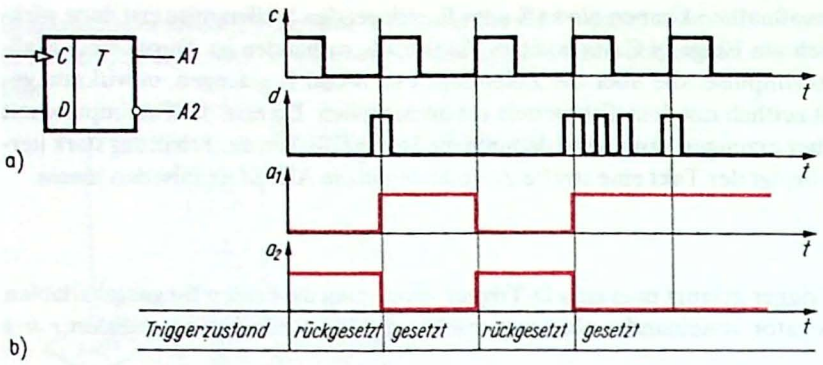


Bild 2.80. D-Trigger mit Taktflankensteuerung
a) Schaltsymbol; b) Zeitdiagramme

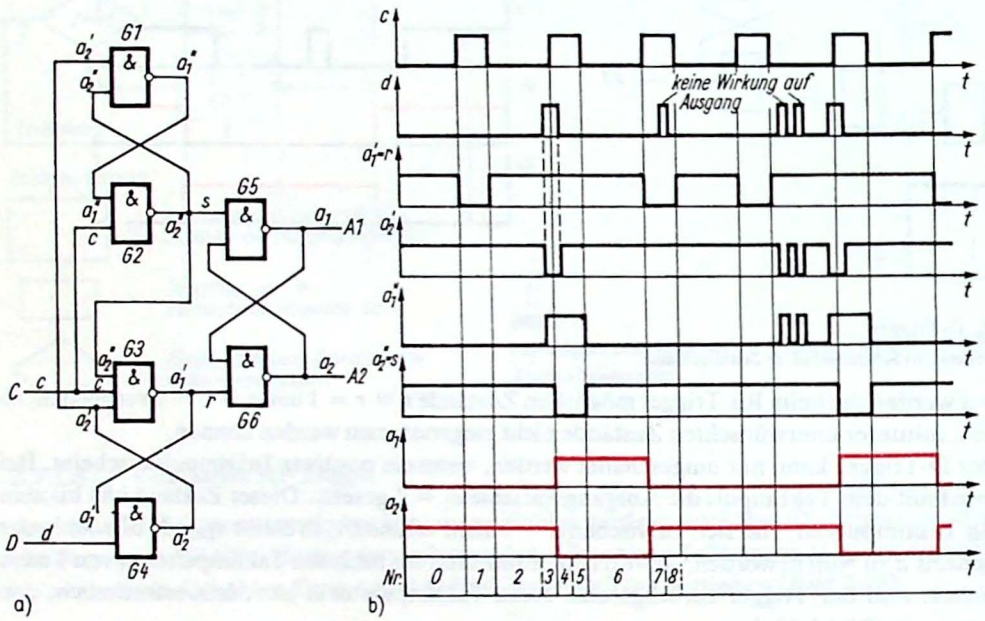


Bild 2.81. D-Trigger mit Taktflankensteuerung
a) Signalflußbild; b) ausführliche Zeitdiagramme

Tafel 2.45. Zustandsfolgetabelle für den D-Trigger mit Taktflankensteuerung nach Bild 2.81

Nr.	d	c	$a_1' = r$	a_2'	a_1''	$a_2'' = s$	a_1	a_2
0	0	0	1	1	0	1	0	1
1	0	1	0	1	0	1	0	1
2	0	0	1	1	0	1	0	1
3	1	0	1	0	1	1	0	1
4	1	1	1	0	1	0	1	0
5	0	1	1	1	1	0	1	0
6	0	0	1	1	0	1	1	0
7	0	1	0	1	0	1	0	1
8	1	1	0	1	0	1	0	1

Das Signalflußbild des taktflankengesteuerten D-Triggers ist im **Bild 2.81a** dargestellt. Bild 2.81 b zeigt ein Zeitdiagramm, und **Tafel 2.45** stellt die Variablenbelegungen in einigen Zuständen des Zeitdiagramms tabellarisch dar.
Das Schalten während der 0/1-Taktflanke wird durch zwei RS-Trigger (bestehend aus G1, G2 und G3, G4, im folgenden Eingangstrigger genannt) erreicht, die vor den Ausgangstrigger G5, G6 geschaltet und gegenseitig verkoppelt sind. Die Taktvariable c wirkt für den oberen und den unteren Trigger als Setzvariable. Zusätzlich wird beim unteren Trigger noch a_2'' als Eingangsvariable wirksam. Die Variable d stellt für den unteren Trigger die Rücksetzvariable dar. Über a_2' wirkt sie auch als Rücksetzvariable für den oberen Trigger.
Bei $d = 0$ soll beim 0/1-Übergang der Taktflanke $a_1 = 0$ folgen bzw., falls dieser Zustand schon vorhanden ist, bleiben. Betrachten wir von diesem Zustand ausgehend den Trigger: $a_1 = 0$ erfordert $a_2 = 1$ und $s = 1$.
Damit folgt:

$$a_2'' = s = 1.$$

Mit $d = 0$ folgt

$$a_2' = 1,$$

und mit

$$a_2'' = 1 \quad \text{wird} \quad a_1'' = 0.$$

Mit $c = 0$ folgt

$$a_1' = r = 1.$$

Damit ergibt sich folgende Lage:

$$s = r = 1$$

sowie

$$a_1 = 0$$

und

$$a_2 = 1.$$

Der obere Eingangstrigger ist gesetzt; der untere Eingangstrigger befindet sich in dem sonst nicht gewünschten Zustand, in dem beide Ausgangsvariablen $a_1' = a_2' = 1$ sind.
Bleibt $d = 0$ und erscheint ein Taktimpuls, so geht zwar a_1' wegen $c = a_2' = a_2'' = 1$ in den Zustand $a_1' = 0$ über, aber der Ausgangstrigger kann nicht schalten, da a_2 bereits im Zustand 1 ist.
Wird aber $d = 1$, bevor die 0/1-Flanke des nächsten Taktimpulses kommt, so ergeben sich folgende Änderungen:
Es wird

$$a_2' = 0 \rightarrow a_1'' = 1.$$

Damit geht der obere Eingangstrigger in den Zustand über, in dem beide Ausgangsvariablen $a_1'' = a_2'' = 1$ sind. Kommt nun die 0/1-Flanke des Taktimpulses, so wird bei $c = 1$

$$a_2'' = 0,$$

gefolgt von

$$a_1 = 1 \quad \text{und} \quad a_2 = 0.$$

Der Ausgangstrigger wurde gesetzt.
Sollte d in der Phase $c = 1$ schon wieder den Zustand $d = 0$ annehmen, passiert nichts, da mit $a_2' = 1$ wegen $a_2'' = 0$ die beiden Eingangstrigger ihre Ausgangszustände beibehalten. Der takt-

flankengesteuerte D-Trigger schaltet also tatsächlich nur in der kurzen Zeit, in der die 0/1-Flanke des Taktimpulses wirksam ist. Es wird nur der Zustand des D-Eingangs ausgewertet, der in diesem Augenblick vorhanden ist. Tafel 2.45 soll den beschriebenen Vorgang in Tabellenform erläutern (Zeilen 1 bis 6). In den Zeilen 7 bis 8 sind die Zustände dargestellt, die beim Rücksetzen des Ausgangstriggers auftreten. Die 8. Zeile zeigt noch einmal, daß der 0/1-Wechsel an D nichts ausmacht, wenn c noch im Zustand 1 geblieben ist.

2.2.1.8. JK-Master-Slave-Trigger

Dieser Trigger besteht aus zwei aufeinanderfolgenden getakteten RS-Triggern, bei denen Rückführungsleitungen zwischen den Ausgängen der letzten und den Eingängen der ersten Stufe existieren. Der erste Trigger wird als Master (Herr), der zweite als Slave (Sklave) bezeichnet. Die Eingangsvariablen heißen bei dieser Triggerschaltung nicht s und r, sondern j und k. Ähnlich wie ein getakteter RS-Trigger behält er den Zustand, in den er zuletzt gebracht wurde, so lange bei, bis er durch den Wechsel der Eingangsvariablen, die den letzten Zustand gerade nicht herbeigeführt hat, in den entgegengesetzten Zustand gesteuert wird **(Bild 2.82)**.

Hat ein positiver Impuls an J die Ausgangsvariable a_1 in den Zustand $a_1 = 1$ gebracht, so haben nachfolgende positive Impulse an J keine Wirkung mehr. Folgt dagegen ein positiver Impuls am Eingang K, so wird bei gleichzeitigem Einwirken des Taktimpulses der entgegengesetzte Zustand am Ausgang eingenommen ($a_1 = 0, a_2 = 1$).

Eine weitere Besonderheit des JK-Master-Slave-Triggers besteht darin, daß im Verlauf eines positiven Taktimpulses die Steuerinformation zunächst vom Master übernommen wird und erst bei der

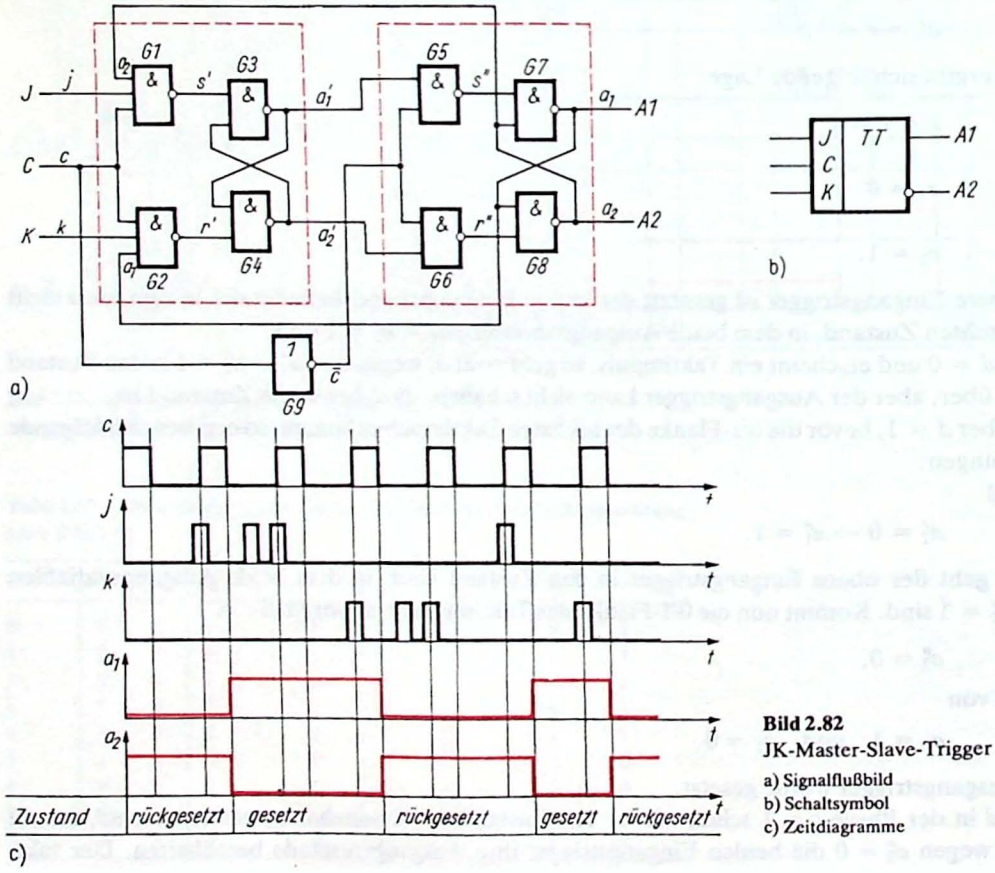


Bild 2.82
JK-Master-Slave-Trigger
a) Signalflußbild
b) Schaltsymbol
c) Zeitdiagramme

1/0-Flanke des Taktimpulses die Information in den Slave gelangt. Dadurch wird zwar die Informationsspeicherung erst nach der vollen Taktimpulsbreite beendet, aber diese zeitliche Verzögerung ist bei einigen Anwendungsfällen sehr erwünscht. Bei dieser Schaltung kann der Zustand, daß sich beide Ausgänge eines Triggers gleichzeitig im Zustand 1 befinden, nicht eintreten.
Erklären wir uns nun die Wirkungsweise des JK-Master-Slave-Triggers anhand der Schaltung nach Bild 2.82a und anhand des Zeitdiagramms nach **Bild 2.83** etwas genauer.

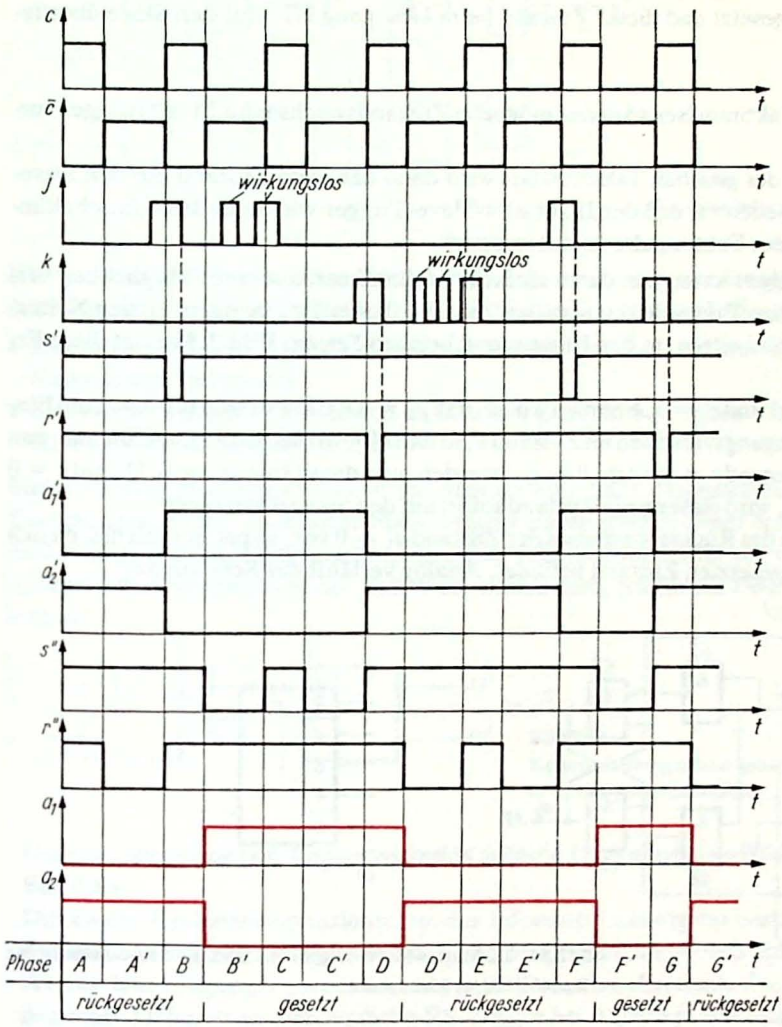


Bild 2.83
JK-Master-Slave-Trigger, ausführliches Zeitdiagramm

Es liege folgender Ausgangszustand (Phase A) vor:

$$a'_1 = a_1 = 0, \qquad a'_2 = a_2 = 1, \qquad s' = s'' = r' = r'' = 1.$$

Während der Phase A′ erscheint am J-Eingang der Zustand $j = 1$, der noch kurzzeitig während der Phase B andauern soll. Mit $c = j = a_2 = 1$ wird dann $s' = 0$, und mit diesem 1/0-Wechsel von s' wird der Mastertrigger gesetzt ($a'_1 = 1, a'_2 = 0$).
Geht nun anschließend der Taktimpuls in den Zustand $c = 0$ so folgt durch G9 $\bar{c} = 1$, und damit wird mit $a'_1 = \bar{c} = 1$ die Variable s'' in den Zustand $'' = 0$ gebracht. Damit wird der Slave gesetzt. Es wird $a_1 = 1$ und $a_2 = 0$.
Die in der Phase B′ folgenden beiden j-Impulse bleiben wirkungslos, weil die Rückführungsleitung vom Ausgang A2 auf das Gatter G1 mit $a_2 = 0$ belegt ist und somit $s' = 1$ bleiben muß. Auch die

Schwankungen der Variablen s'' in den Phasen B′ und C′ (wegen $\bar{c} = a_1' = 1$) bleiben wirkungslos, weil der Slave bereits gesetzt ist.

Erst der in der Phase C′ am Eingang K auftretende Impuls führt in der Phase D mit $c = k = a_1 = 1$ zum Umschalten des Masters in den rückgesetzten Zustand. Im Übergang von Phase D zur Phase D′ wird dann diese Rücksetzlage auf den Slave übertragen. Auch die in der Phase D′ folgenden k-Impulse haben wieder keine Wirkung. Erst beim Übergang von Phase E′ zur Phase F wird der Master durch den j-Impuls wieder gesetzt und dieser Zustand beim Übergang F/F′ auf den Slave übertragen.

Ergebnis:

- Während des positiven Taktimpulses wird der mögliche Zustandswechsel im Mastertrigger vorgenommen.
- Während der 1/0-Flanke des gleichen Taktimpulses wird dann der neue Zustand auf den Slavetrigger übertragen. Das bedeutet, daß der JK-Master-Slave-Trigger wirksame J/K-Umschaltimpulse mit der 1/0-Flanke des Taktimpulses synchronisiert.

Einen JK-Master-Slave-Trigger kann man durch einfache Maßnahmen mit einer Möglichkeit versehen, Master und Slave ohne Takteinwirkung in den Zustand 0 oder 1 zu bringen. In den Nullzustand bringen nennt man *Rücksetzen*, in den Einszustand bringen *Setzen*. **Bild 2.84** zeigt diese Erweiterungsmaßnahmen.

Im Zustand $c = 0$ sind $s' = 1$ und $r' = 1$. Nehmen wir an, daß $a_1' = 1$ ist. Dann sind mit $r = 1$ am Eingang des Gatters G4 alle Eingangsvariablen im Zustand 1, so daß $a_2' = 0$ bleiben kann. Gibt man nun aber an R kurzzeitig den Zustand $r = 0$, so muß $a_2' = 1$ werden, und damit folgt $a_1' = 0$. Da mit $c = 0$ der Zustand $\bar{c} = 1$ herrscht, wird dieser neue Zustand sofort auf den Slave übertragen.

Liegt bereits vor Eintreffen des Rücksetzimpulses der Zustand $a_1' = 0$ vor, so passiert nichts, da sich der Trigger bereits im rückgesetzten Zustand befindet. Analog verläuft der Setzvorgang.

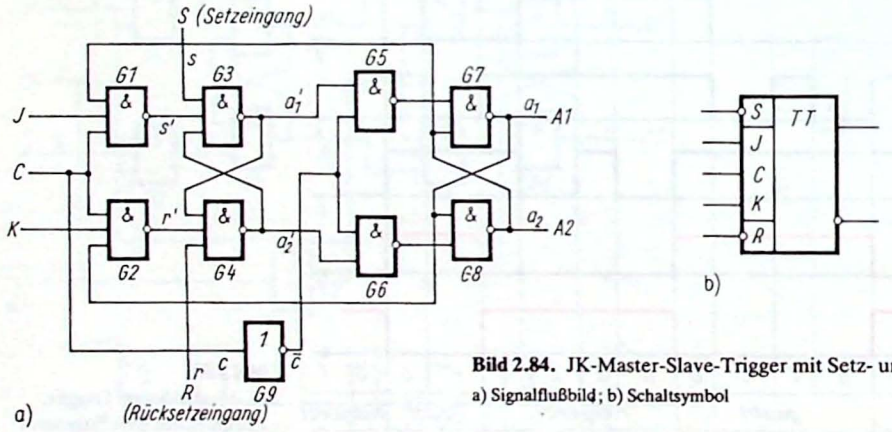


Bild 2.84. JK-Master-Slave-Trigger mit Setz- und Rücksetzeingang
a) Signalflußbild; b) Schaltsymbol

2.2.2. Zusammenschaltung von Triggern

Der einzelne Trigger kann als Elementarspeicherzelle angesehen werden.
Zusammenschaltungen von Elementarspeicherzellen ergeben Speicherschaltungen.
Durch Hinzufügen einer Steuerlogik kann man zu Automaten gelangen. Bereits Schaltungen mit wenigen Triggern und einer entsprechenden Steuerlogik ergeben schon Automaten mit recht beachtlichen Fähigkeiten.
Je größer die Anzahl der verwendeten Glieder wird, um so mehr muß man darauf achten, daß die entworfene Struktur auch informationstechnisch effektiv und übersichtlich gestaltet ist.

Doch beginnen wir zunächst mit den grundlegenden Betrachtungen. Die Übertragung von Triggerspeicherinhalten kann nach verschiedenen Varianten erfolgen. **Bild 2.85** zeigt die erste Variante in Form der Kettenschaltung zweier getakteter RS-Trigger.

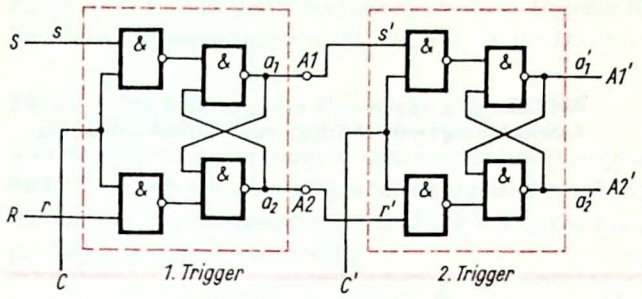


Bild 2.85
Kettenschaltung zweier getakteter RS-Trigger

Die an A1 und A2 stehenden Informationen ($a_1 = \bar{a}_2 = x$) werden dann in den zweiten Trigger übernommen, wenn am Takteingang C' ein 0/1-Sprung erfolgt. Dabei gibt es zwei Möglichkeiten:

- Der zweite Trigger befindet sich bereits in dem Zustand, den der erste Trigger übertragen soll. Dann ändert sich nichts.
- Der zweite Trigger befindet sich im inversen Zustand zum ersten Trigger. Dann wird er mit der 0/1-Flanke an C' umgeschaltet.

Bild 2.85 zeigt starke Ähnlichkeit mit dem im Bild 2.82a dargestellten JK-Master-Slave-Trigger. Tatsächlich stellt der JK-Master-Slave-Trigger ein Beispiel zu dieser ersten Variante dar. Er hat jedoch zusätzliche Rückführungsleitungen und weist den Sonderfall auf, daß der Slave unbedingt die Information des Masters zu der Zeit übernehmen muß, bei der die 1/0-Flanke des Taktimpulses erscheint.

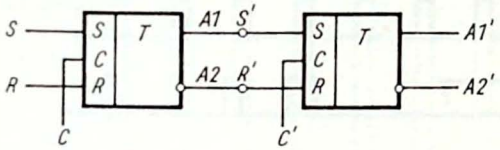


Bild 2.86
Kettenschaltung zweier getakteter RS-Trigger in Schaltsymboldarstellung

Unter Anwendung von Triggersymbolen geht die Darstellung des Bildes 2.85 über in diejenige von **Bild 2.86**.

Die zweite Realisierungsvariante bei der Informationsübergabe besteht darin, vor der Übergabe den zweiten Trigger zurückzusetzen. Dann benötigt man nur noch eine Verbindung des Ausgangs A1 mit dem Eingang S', weil dann nur noch der evtl. auftretende Zustand $a_1 = 1$ übertragen werden muß. Dabei kann man einfache RS-Trigger bei Aufbau je einer zusätzlichen Torschaltung verwenden.

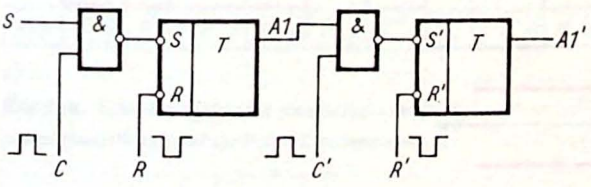


Bild 2.87
Kettenschaltung zweier RS-Trigger

Der Nachteil dieser im **Bild 2.87** dargestellten Variante besteht darin, daß zur Informationsübergabe zwei zeitlich nacheinander folgende Impulse benötigt werden, so daß die Übernahmezeitdauer vergrößert wird.

Die in der Kopplung einfachste Variante erhält man bei der Anwendung von D-Triggern **(Bild 2.88)**. Es werden nur eine Kopplungsleitung und ein Übernahmeimpuls benötigt.
Zu beachten ist, daß bei allen Verfahren durch die Übertragung der Information an den Empfängertrigger die Information im Sendetrigger nicht verlorengeht. Es wird praktisch nur eine Kopie vorgenommen.

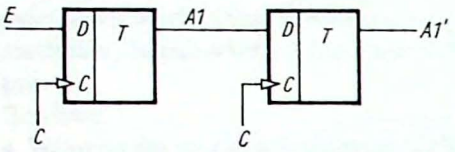


Bild 2.88
Kettenschaltung zweier D-Trigger mit Taktflankensteuerung

2.2.3. Register

Ein Register ist eigentlich ein Verzeichnis oder eine Tabelle. Im Sinne der Informationsverarbeitung ist es eine kleine Speicheranordnung für eine nicht allzu große Anzahl von Bitstellen. Register dienen meist nur zur kurzzeitigen Speicherung von Informationen, können aber auch bestimmte Operationen, wie Stellenverschiebung, Zählung, Serien-Parallel- oder Parallel-Serienwandlung von Binärwörtern, realisieren.

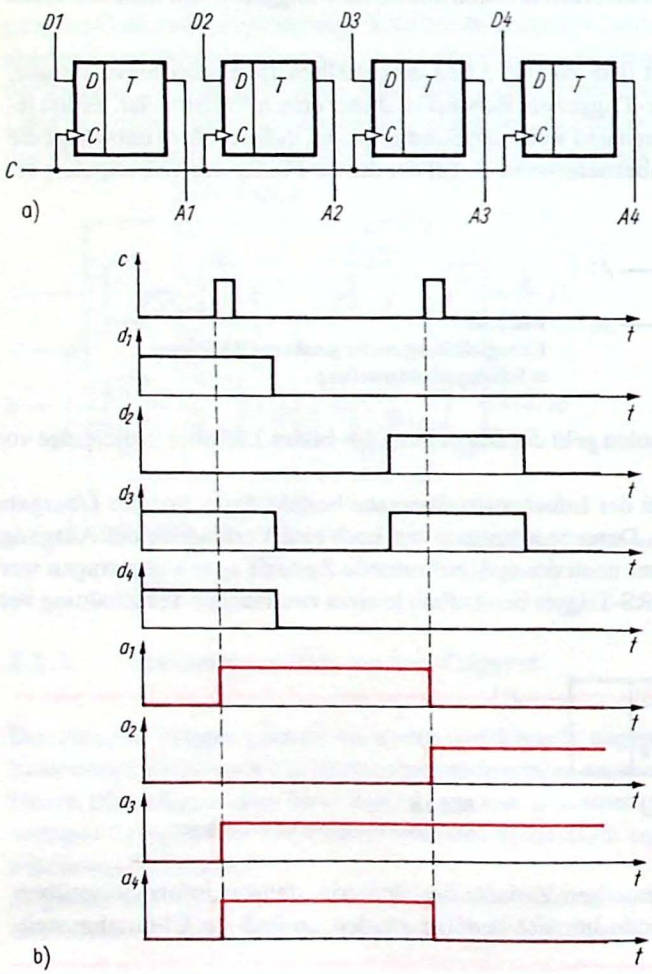


Bild 2.89. Register mit Paralleleingängen
a) Signalflußbild; b) Zeitdiagramme

2.2.3.1. Parallelregister

Parallelregister dienen zur parallelen Informationsübernahme von Mehrbitdatenwörtern. Sie werden der vorgegebenen Wortbreite angepaßt, z. B. für 4, 8, 16, 32 Bit. Im **Bild 2.89** sind ein 4-Bit-Parallelregister aus taktflankengesteuerten D-Triggern und ein Zeitdiagrammbeispiel dargestellt. Die Dateneingänge sind mit D1, D2, D3, D4 die Datenausgänge mit A1, A2, A3, A4 bezeichnet.

2.2.3.2. Schieberegister für serielle Eingabe

Als erste Registeranordnung wollen wir das Schieberegister für *eine Schieberichtung* kennenlernen. **Bild 2.90 a** zeigt ein vierstelliges Schieberegister mit JK-Master-Slave-Triggern, das noch mit einer Rücksetzleitung ausgestattet ist. Bild 2.90 b zeigt das Schaltungssymbol, und Bild 2.90 c gibt ein zugehöriges Zeitdiagramm wieder.

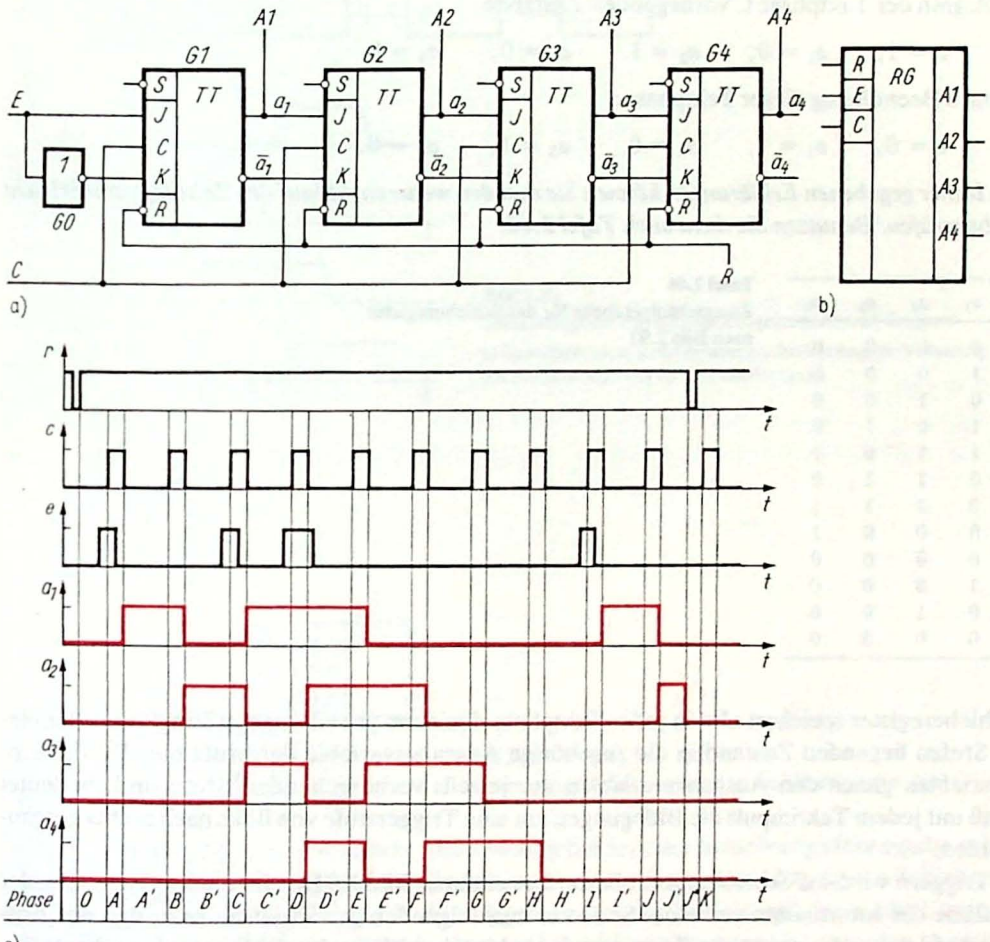


Bild 2.90. Schieberegister für eine Schieberichtung
a) Signalflußbild; b) Schaltsymbol; c) Zeitdiagramme

Das Zeitdiagramm beginnt mit dem Ereignis, daß an den Rücksetzeingang R vorübergehend $r = 0$ gegeben wird. Damit werden alle vier Trigger gleichzeitig in die Rücksetzlage gebracht. Es besteht dann der Anfangszustand $a_1 = a_2 = a_3 = a_4 = 0$. Dieses Rücksetzen wirkt sich so aus, als ob zuletzt an den Eingängen K jeweils ein positiver Impuls aufgetreten wäre.

Zur 0/1-Flanke des Taktimpulses der Phase A sei ein Eingangsimpuls $e = 1$ vorhanden. Dann wird mit der 1/0-Flanke dieses Taktimpulses diese Information mit $a_1 = 1$ in den Slave des Triggers G1 eingespeichert. Auch die Trigger G2 bis G4 erhalten den Taktimpuls zugeführt. Da aber während der Zeitdauer dieses Impulses an ihren J-Eingängen der Zustand $j = 0$ herrscht, ändert sich bei ihnen nichts.
Zur Taktphase B ist $e = 0$. Dann wird über den Negator G_0 der Zustand $k = 1$ in den Trigger G1 eingegeben. Damit wird am Ende der Phase B G1 rückgesetzt ($a_1 = 0$). Während der Taktphase B ist $a_1 = 1$. Das bedeutet, daß der an Trigger G2 ebenfalls eingegebene Taktimpuls die Information $a_1 = 1$ mit der 0/1-Flanke in den Master und mit der 1/0-Flanke in den Slave des Triggers G2 einschreibt. Damit besteht folgender Zustand in der Phase B′:

$$a_1 = 0, \quad a_2 = 1, \quad a_3 = 0, \quad a_4 = 0.$$

Die zu Beginn der Taktphase C vorliegenden Zustände

$$e = 1, \quad a_1 = 0, \quad a_2 = 1, \quad a_3 = 0, \quad a_4 = 0$$

lauten nach Beendigung dieser Taktphase:

$$e = 0, \quad a_1 = 1, \quad a_2 = 0, \quad a_3 = 1, \quad a_4 = 0.$$

Mit den bisher gegebenen Erklärungen können Sie nun den weiteren Ablauf des Zeitdiagramms leicht selbst überprüfen. Benutzen Sie dazu auch ***Tafel 2.46****.*

Tafel 2.46
Zustandsfolgetabelle für das Schieberegister nach Bild 2.90

Phase	a_1	a_2	a_3	a_4
0	0	0	0	0
A′	1	0	0	0
B′	0	1	0	0
C′	1	0	1	0
D′	1	1	0	1
E′	0	1	1	0
F′	0	0	1	1
G′	0	0	0	1
H′	0	0	0	0
I′	1	0	0	0
J′	0	1	0	0
J″	0	0	0	0

Das Schieberegister speichert also in jeder Taktphase den dann jeweils an den Eingängen der einzelnen Stufen liegenden Zustand in die zugehörige Ausgangsvariable der Stufe ein. Da die Eingangsvariablen gleich den Ausgangsvariablen der jeweils vorhergehenden Stufe sind, bedeutet dies, daß mit jedem Taktimpuls die Bitlegungen um eine Triggerstufe von links nach rechts geschoben werden.
Mit D-Triggern wird das Schieberegister besonders einfach **(Bild 2.91a)**. Es muß jedoch bei jeder Übernahme der Information von einer Stufe zur nachfolgenden gewährleistet sein, daß der Ausgangszustand des vorhergehenden Triggers so lange bestehen bleibt, bis die Übernahme dieses Zustands durch den nachfolgenden Trigger beendet ist. Das kann erreicht werden, indem man die Abfallverzögerung des 1/0-Übergangs entsprechend größer macht als die Anstiegsverzögerung beim 0/1-Übergang (Bild 2.91b). Wenn diese Bedingung nicht realisiert werden kann, muß man zur Anwendung zweistufiger Trigger (z. B. JK-Master-Slave-Trigger) übergehen.
Welche Anwendungsmöglichkeiten bietet nun ein solches Schieberegister?
Zunächst kann es als *Auffangspeicher* für eine serielle (nacheinanderfolgende) Dateneingabe einzelner Binärschritte benutzt werden. So kann man z. B. in das im Bild 2.90a dargestellte Schiebere-

gister 4 Bit einspeichern. Die eingespeicherte Bitbelegung kann später wiederum *seriell*, aber auch *parallel* ausgegeben werden.

Zur seriellen Ausgabe wird der Empfänger des Speicherinhalts an den Ausgang A4 angeschlossen **(Bild 2.92)**. Versetzt man den Enable-Eingang *e* des Tores G3 in den Zustand 1, so übernimmt bei Erscheinen des Taktimpulses der Trigger G2 jeweils den an A4 auftretenden Zustand. Die Funktion des Schieberegisters besteht in diesem Fall als *Zwischenspeicher* oder auch *Verzögerungsglied*. Die minimale Verzögerung beträgt vier Taktzeiten. Die maximal mögliche Verzögerung hängt davon ab, wann die Übernahmetakte ausgelöst werden.

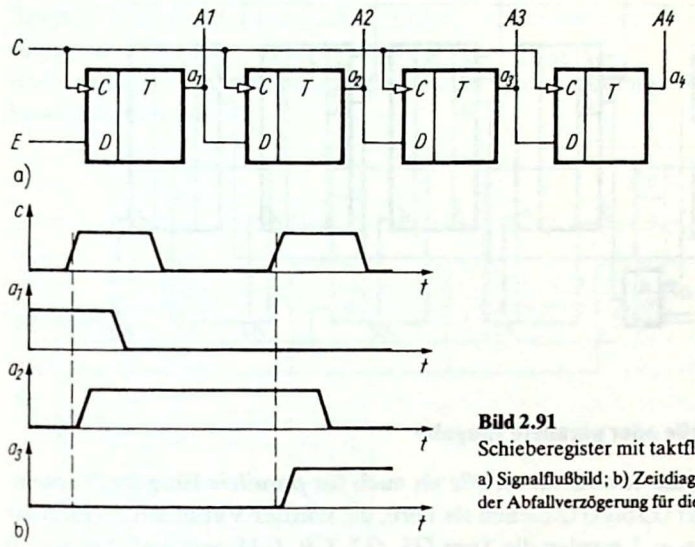


Bild 2.91
Schieberegister mit taktflankengesteuerten D-Triggern
a) Signalflußbild; b) Zeitdiagramm zur Erläuterung der Bedeutung der Abfallverzögerung für die Funktionsweise

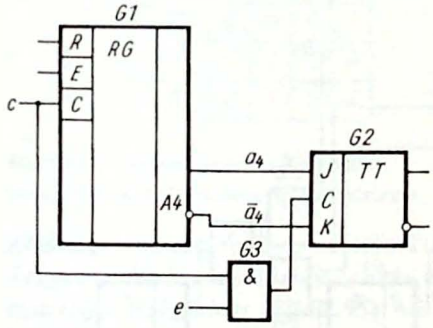


Bild 2.92
Anwendung des Schieberegisters als Verzögerungsglied

Eine sehr interessante und nützliche Anwendung bei serieller Schieberegisterausgabe erhält man durch den im **Bild 2.93** dargestellten *Busbeobachter*. Durch die Anschaltung von Schieberegistern mit einer größeren Triggerzahl (z. B. 64 Triggern) an je eine Busleitung können die Kopien von 64 aufeinanderfolgenden Bit dieser beobachteten Datenleitung eingespeichert werden. Anschließend kann man die *eingefangenen* Bit der Reihe nach in aller Ruhe aus dem Schieberegister herausholen und dabei kontrollieren, ob der Prozeß ordnungsgemäß verlaufen ist. Während des normalen Datenverkehrs ist eine solche Kontrolle durch den Menschen nicht möglich, da die Daten aus ökonomischen Gründen mit hoher Geschwindigkeit übertragen werden.

Natürlich muß man das Problem klären, wie man aus dem schnellen Datenstrom gerade jeweils diejenigen Bit herausholt, die möglicherweise Aufschluß über eine fehlerhafte Reaktion geben. Das läßt sich dadurch erreichen, indem man eine bestimmte *Bitstruktur* vorgibt, von der ab eingespei-

chert werden soll. Eine Vergleichsschaltung (s. S. 65) bildet dann den Auslöse- oder Startimpuls. Ein Zähler, der von 1 bis 64 zählt, stoppt die Einspeicherung.
Solche Kontrollgeräte, die außerdem noch weitere Auswerte- und Darstellungsmöglichkeiten aufweisen, nennt man *Logikanalysatoren*.

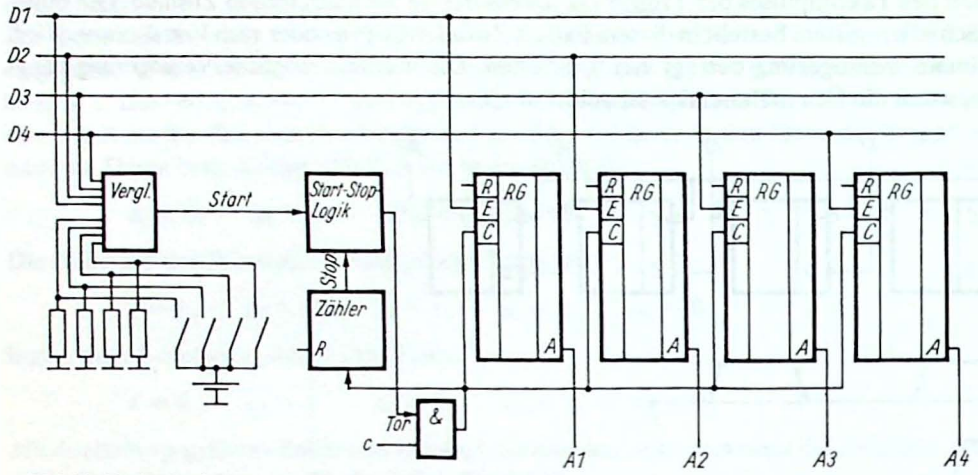


Bild 2.93. Busbeobachter

2.2.3.3. Schieberegister für serielle oder parallele Eingabe

Bild 2.94 zeigt ein Schieberegister, das sowohl für *serielle* als auch für *parallele* Eingabe Verwendung finden kann. Die UND-Glieder G5 bis G12 dienen als Tore, die von der Variablen s_1 gesteuert (oder programmiert) werden. Bei $s_1 = 1$ werden die Tore G5, G7, G9, G11 geöffnet; bei $s_1 = 0$ werden über den Negator G14 die Tore G6, G8, G10, G12 auf Durchlaß geschaltet.

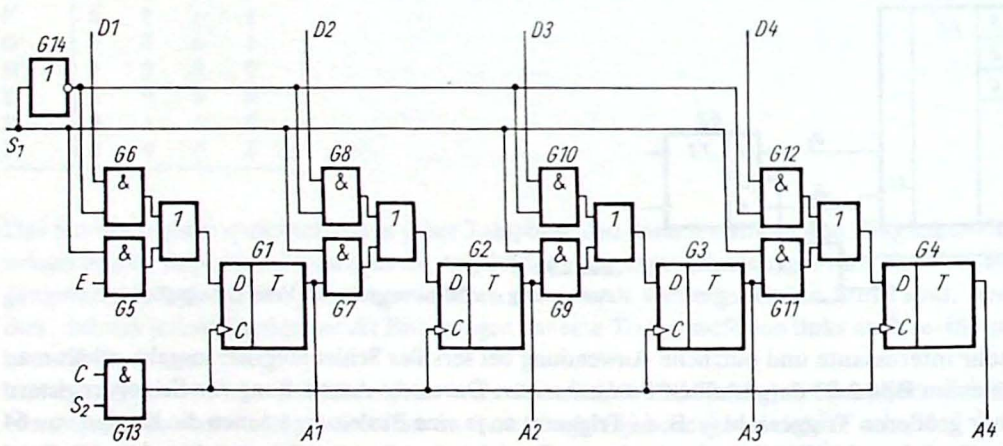


Bild 2.94. Schieberegister für serielle oder parallele Dateneingabe

Ist $s_2 = 1$, so ist der Takt freigegeben. Ist dabei $s_1 = 0$, so werden die Daten D1 bis D4 mit der 0/1-Flanke des Taktimpulses in die Trigger G1 bis G4 eingespeichert und liegen dann an den Ausgängen A1 bis A4.
Wird bei $s_2 = 1$ auch $s_1 = 1$, so folgen mit der 0/1-Taktflanke eine serielle Eingabe und die Verschiebung der gespeicherten Bit um jeweils eine Stelle.

2.2.3.4. Weitere Anwendungsbeispiele für Schieberegister

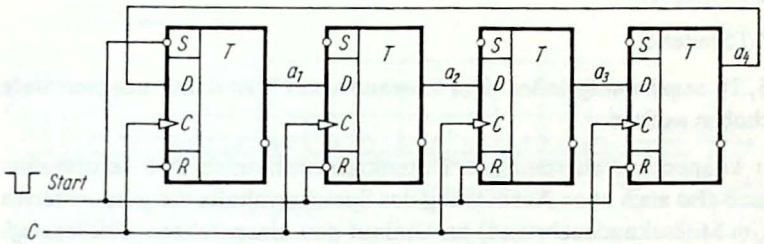
Division. Schiebt man eine Dualzahl um eine Stelle nach rechts, so entspricht das der Division durch die Zahl 2.

Beispiel: Eingabe der Dualzahl 01110 ≙ 14, Verschieben nach rechts mit Auffüllung einer Null von links ergibt 00111 ≙ 7. Bei n-stelliger Verschiebung nach rechts und der Auffüllung mit Nullen von links erhält man eine Division durch 2^n.

Multiplikation. Schiebt man eine Dualzahl um eine Stelle nach links, so entspricht das der Multiplikation mit der Zahl 2.

Beispiel: Eingabe der Dualzahl 01110 ≙ 14, Verschieben um eine Stelle nach links mit Auffüllung einer Null von rechts ergibt 11100 ≙ 28.

Bei n-stelliger Verschiebung nach links und der Auffüllung mit Nullen von rechts erhält man eine Multiplikation mit 2^n.



a)

Nr.	Start	Takt C	a_1	a_2	a_3	a_4
1	⊓	—	1	0	0	0
2	—	⊓	0	1	0	0
3	—	⊓	0	0	1	0
4	—	⊓	0	0	0	1
5	—	⊓	1	0	0	0
6	—	⊓	0	1	0	0

b)

Bild 2.95. Schieberegister als Ringzähler
a) Signalflußbild; b) Darstellung der Zustandsfolgen

Zählung. Versetzt man den ersten Trigger eines Schieberegisters in den Zustand 1, alle anderen Trigger in den Zustand 0, und schiebt man diesen Einszustand zyklisch durch das Register, so erhält man einen *Ringzähler* **(Bild 2.95)**. Mit jedem Taktimpuls läuft der Einszustand um eine Stufe weiter. Aus der Stellung derjenigen Stufe, die gerade den Einszustand aufweist, kann man auf die Anzahl der bisher eingetroffenen Taktimpulse schließen.

Die Arbeitsweise eines Ringzählers kann auch zur Nachbildung eines rotierenden Schalters ausgenutzt werden.

Beispiel: Schließt man an jedem Ausgang einer zum Ringzähler geschalteten Triggerkette einen Auslöseschalter für einen bestimmten Teilschritt eines Prozesses an, so kann man diese Prozeßschritte zeitverzögert in der vorgegebenen Reihenfolge ablaufen lassen.

Parallel-Serien-Wandler. Gibt man die Daten parallel in das Schieberegister ein und erfolgt die Ausgabe seriell, so kann man jedes Bit eines Wortes nacheinander am Ausgang empfangen. Damit ist die *Umsetzung* einer Parallelübertragung in eine serielle Übertragung gegeben.

Serien-Parallel-Wandler. Gibt man dagegen die Daten seriell ein, so kann man alle Bit eines Datenworts zunächst der Reihe nach einschieben und sie danach auf einmal parallel weiterreichen.

2.2.3.5. Andere Ausführungsformen von Schieberegistern

Bei unipolaren Transistoren (MOS-Technik) hat man so hohe Eingangswiderstände realisiert, daß man mit den sonst schädlichen Schalt- und Transistorkapazitäten nützliche Anwendungen mit Hilfe der Ladungsspeicherung über einige Millisekunden erzielen kann. Man kann unter Ausnutzung dieses Speichereffekts dynamische Schieberegister aufbauen **(Bild 2.96)**. Zur Diskussion der Schaltung nach Bild 2.96 wollen wir folgende Ausgangslage annehmen:

C1 ist negativ geladen → T1 ist leitend ≙ Einszustand.

Es vollzieht sich dann folgender Vorgang:

Takt 1: $c_1 = 1 \rightarrow$ T2 und T3 leitend. Da jedoch T1 niederohmiger als T2 ist, geht u_{C2} gegen 0 V → T4 gesperrt.

Takt 2: $c_1 = 0 \rightarrow$ T2 und T3 gesperrt.

$c_2 = 1 \rightarrow$ T5 und T6 leitend.

C1′ wird über T5, T6 negativ aufgeladen. Der Zustand von C1 ist damit um eine Stufe nach rechts verschoben worden.

Dynamische Schieberegister können nur im ständigen Datenkreislaufbetrieb ihre Information gespeichert behalten. Man muß also auch ohne Ausnutzung des Speicherinhalts die gespeicherten Zustände von Zeit zu Zeit (im Millisekundenabstand) im Umlauf neu einspeichern. Dieses *Auffrischen* (refresh) wird aber im Zeitraum von Mikrosekunden realisiert, so daß noch genügend Zeit zum normalen Betrieb bleibt.

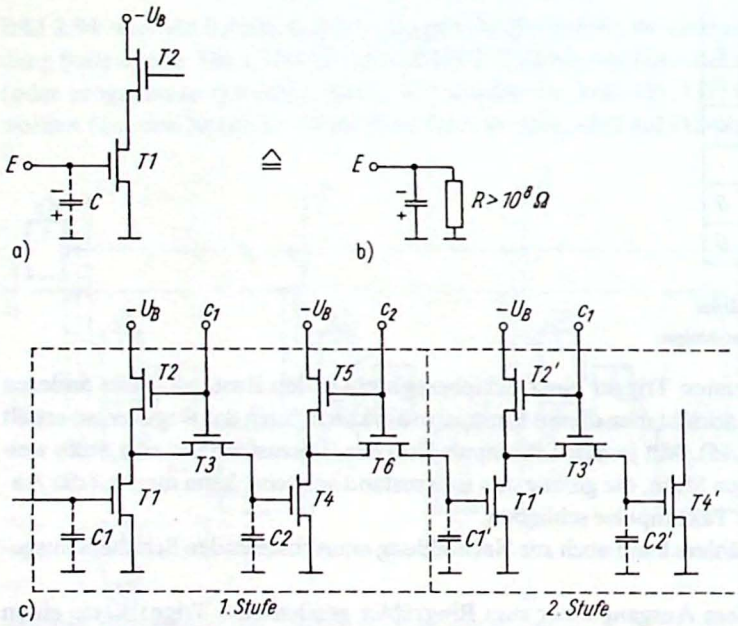


Bild 2.96. Dynamisches Schieberegister in MOS-Technik
a) Eingang zur 1. Stufe; b) Ersatzschaltung; c) Schaltung

Der Vorteil der dynamischen Schieberegister liegt in der möglichen hohen Packungsdichte, d. h. im geringen Platzbedarf. Eine noch rationellere Ausführungsform eines Schieberegisters ist in Form einer *CCD-Anordnung* (CCD ≙ charge coupled device, ladungsgekoppelte Vorrichtung) entwikkelt worden.

Die Vorteile der CCD-Anordnung liegen im geringen Leistungsbedarf, in der höheren Packungsdichte, einer besser beherrschbaren Fertigung und in geringeren Herstellungskosten als für MOS-Register.

Die gesamte Anordnung hat einen relativ einfachen Aufbau. Zur einführenden Erklärung soll zunächst **Bild 2.97** dienen.

Ein Block aus n-Silizium ist mit einer Isolierschicht aus Siliziumoxid (SiO_2) abgeschlossen. Auf der SiO_2-Schicht sind gegeneinander isolierte metallische Elektroden angebracht. Legt man alle Elek-

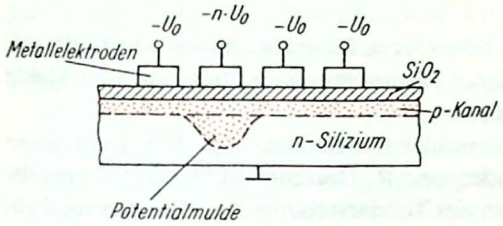


Bild 2.97
Vereinfachte Darstellung eines CCD-Schieberegisters

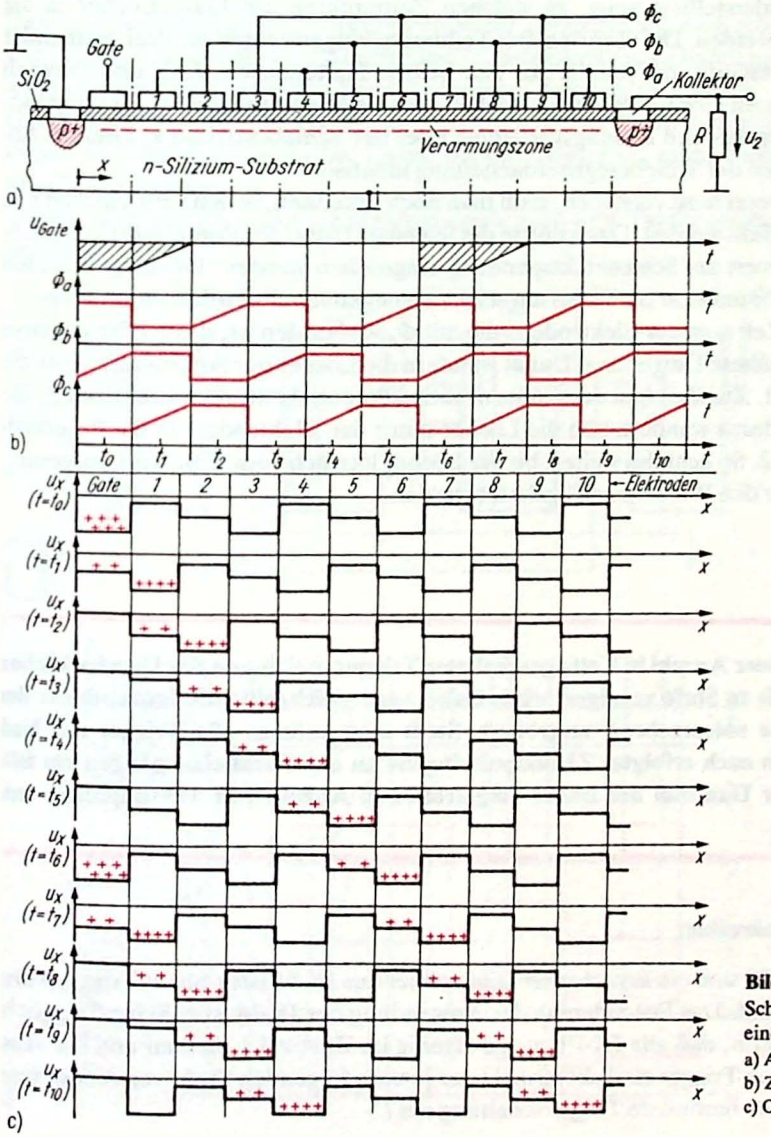


Bild 2.98
Schematische Darstellung eines CCD-Schieberegisters
a) Aufbauskizze
b) Zeitdiagramme
c) Ortsdiagramme

troden an die gleiche negative Spannung – U_0 gegen Masse, so bildet sich an der Grenze zwischen n-Silizium und SiO_2-Schicht eine Verarmungszone an beweglichen Ladungsträgern (Elektronen) aus. Im Laufe der Zeit diffundieren Löcher aus dem n-Silizium in die Verarmungszone hinein, so daß dann ein p-leitender Kanal entsteht.
Macht man nun die Spannung an einer Elektrode stärker negativ ($-n\,U_0$) als an den anderen, so wird sich an dieser Stelle die Verarmungszone verbreitern. Dadurch entsteht eine Potentialmulde. In dieser Potentialmulde kann man positive Ladungsträger (Löcher) für kurze Zeit (maximal 10 µs) festhalten.
Verschiebt man diese Potentialmulde mit Hilfe eines besonderen Dreiphasentakts von Elektrode zu Elektrode, so wandern die in der Mulde festgehaltenen Löcher mit der Verlagerung der Mulde mit. Auf diese Weise werden sie zum Ausgang transportiert.
Bild 2.98 zeigt ein solches Schieberegister mit zehn Verschiebeelektroden, den drei Anschlüssen für die Takte Φ_a, Φ_b, Φ_c und mit einem Auskoppelwiderstand R. Darunter ist ein Zeit-/Orts-Diagramm dargestellt. In diesem Diagramm sind die ersten vier Teildarstellungen von oben reine Zeitdiagramme. Die erste Teildarstellung zeigt, zu welchen Zeitpunkten am Gate Löcher in die Anfangsmulde eingebracht werden. Die nächsten drei Teildarstellungen zeigen die drei zueinander phasenverschobenen Taktverläufe, die jeweils aus den beiden Haltephasen stark und schwach negativ sowie aus einer linearen Übergangsphase bestehen. Die Teildiagramme $u_x = f(x)$ sind Momentdarstellungen der Potential- und Ladungsverteilung über der Schiebestrecke x. Diese x-Abschnitte sind mit den Strecken der Schieberegisterdarstellung identisch.
Um die Wirkung der Schiebetakte zu verstehen, muß man noch erkennen, welche Elektroden von welchem Schiebetakt beeinflußt werden. Dann dürfte die jeweilige Höhe der Potentiallinien in Abhängigkeit vom Momentanwert der Schiebetaktspannung eingesehen werden. Die eingebrachten Löcher haben die Tendenz, immer zur Stelle des am stärksten negativen Potentials zu wandern.
Zum Beispiel herrscht zur Zeit t_1 an der Elektrode 1, die mit Φ_a verbunden ist, das größte negative Potential, bezogen auf die nähere Umgebung. Damit wandern die Löcher der Anfangsmulde in die Mulde unter der Elektrode 1. Zur Zeit t_2 ist das größte negative Potential unter der Elektrode 2, die von Φ_b gesteuert wird. Dadurch wandern nun die Löcher unter der Elektrode 1 in die Potentialmulde unter der Elektrode 2. So geht das weiter, bis die Löcher letztlich vom Kollektor aufgesammelt und als Stromstoß über den Widerstand R geleitet werden.

2.2.4. Binärzähler

Binärzähler bestehen aus einer Anzahl in Kette geschalteter Trigger, bei denen das Durchschieben der Triggerinhalte von Stufe zu Stufe verzögert wird. Dabei wird gleichzeitig die Impulsdauer der einzelnen Ausgangsimpulse entsprechend vergrößert. Stellt man anfangs alle Trigger auf Null (Rücksetzen), so kann man nach erfolgter Zählimpulseingabe an den Parallelausgängen ein Bitmuster abnehmen, das der Dualzahl der bisher eingetroffenen Anzahl von Taktimpulsen entspricht.

2.2.4.1. Asynchrone Binärzähler

Vorwärtszähler. Im **Bild 2.99** sind ein *asynchroner* Binärzähler aus JK-Master-Slave-Triggern und sein Zeitdiagramm dargestellt. Das Besondere in der Anwendung der JK-Master-Slave-Trigger in dieser Schaltung besteht darin, daß alle JK-Eingänge ständig im Zustand 1 bleiben und die Ausgangsvariablen der einzelnen Trigger als Taktvariable der jeweils folgenden Stufe verwendet werden. Wie wirkt sich nun diese besondere Triggerschaltung aus?

Dadurch, daß $j = k = 1$ ist, kann jeder am Takteingang C erscheinende positive Impuls den JK-Master-Slave-Trigger umschalten. *Das können Sie sich selbst anhand des Bildes 2.82 auf S. 100 erklären.*

Sie werden erkennen, daß durch die Rückführungsleitungen von den Ausgängen zu den Eingangs-NAND-Gliedern jeweils dasjenige NAND-Glied einen 1/0-Sprung mit Hilfe des Taktimpulses an seinem Ausgang vollziehen muß, das den Trigger umschalten kann. Befindet sich der Trigger z. B. im rückgesetzten Zustand, so bekommt das obere Eingangs-NAND-Glied den Zustand 1 zurückgeführt, und der nächste Taktimpuls kann den Setzvorgang realisieren.

Wir stellen zusammenfassend fest:

- Sind beim JK-Master-Slave-Trigger alle JK-Eingänge im Zustand 1, so schaltet jeder Taktimpuls den Trigger um. Mit der 0/1-Flanke des Taktimpulses wird der Master umgeschaltet. Die 1/0-Flanke des gleichen Impulses veranlaßt die Übernahme des Masterzustands in den Slave.
  Damit der Trigger in die Ausgangslage zurückkehren kann, werden zwei volle Taktimpulse benötigt.

Damit ist auch die weitere Wirkungsweise der Zählschaltung nach Bild 2.99a zu verstehen.

Der Trigger G1 nimmt jeweils nach zwei Taktimpulsen seine Ursprungslage ein, hat also im Verlauf von zwei Taktimpulsen einen Impuls an seinem Ausgang abgegeben.

Der Trigger G2 gibt seinerseits im Verlauf von zwei Ausgangsimpulsen des Triggers G1 einen Impuls an seinem Ausgang ab. Damit hat der Trigger G2 einen Ausgangsimpuls nach vier Taktimpulsen vollendet.

Trigger G3 vollendet einen Ausgangsimpuls nach acht Taktimpulsen, Trigger G4 nach 16 Taktimpulsen.

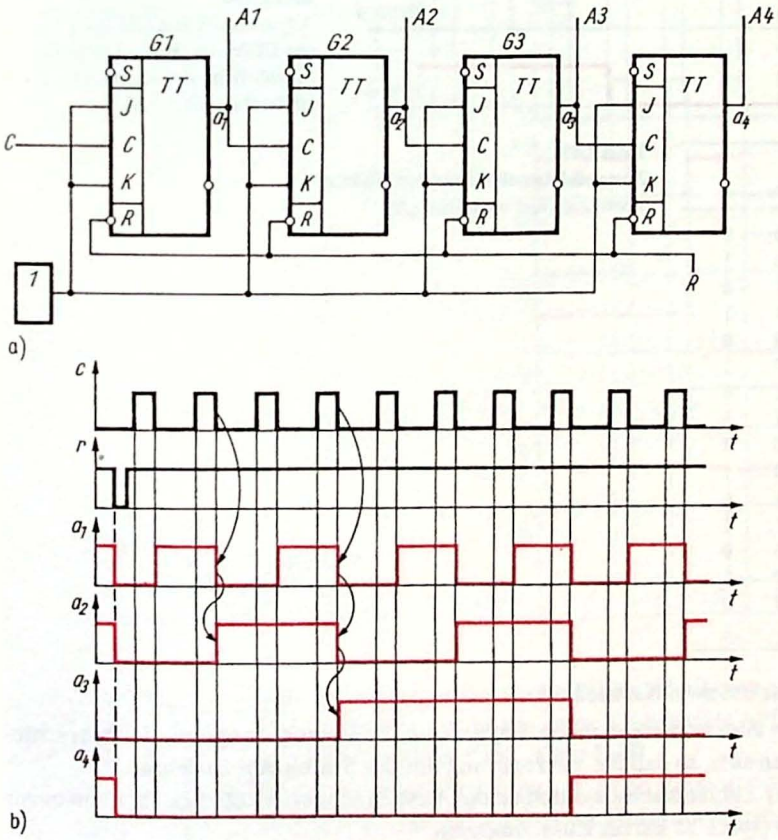


Bild 2.99
Asynchroner Binärzähler aus JK-Master-Slave-Triggern
a) Signalflußbild; b) Zeitdiagramme

Auf diese Weise kommt das Zeitdiagramm im Bild 2.99b zustande. *Beachten Sie, daß immer die 1/0-Flanken der an den Takteingängen zugeführten Impulse den Umschaltpunkt bestimmen.* Im **Bild 2.100** ist das Zeitdiagramm der vier Trigger ausführlicher dargestellt. Verfolgt man die Zustände, die nach den einzelnen Taktimpulsen an den Ausgängen bestehen, erhält man die **Tafel 2.47**.

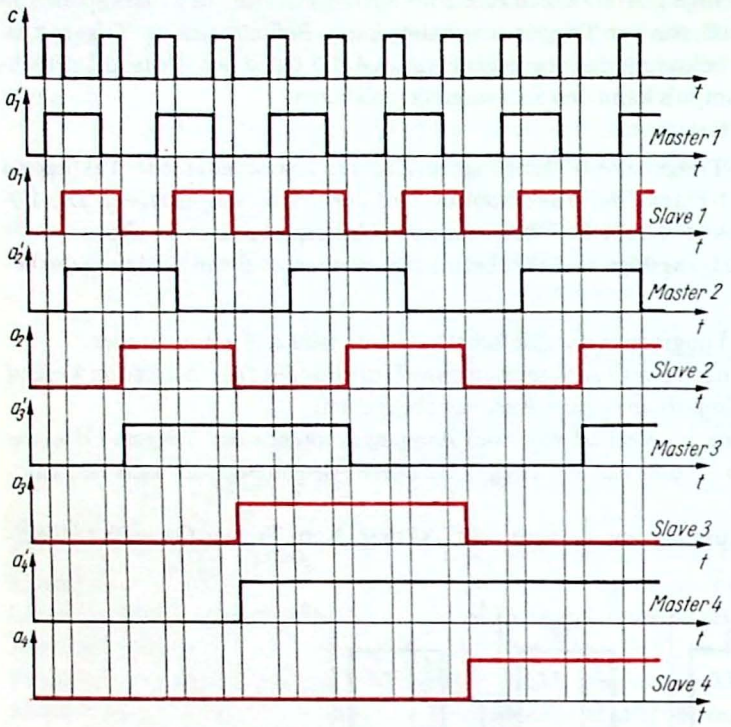


Bild 2.100
Asynchroner Binärzähler aus JK-Mater-Slave-Triggern, ausführliche Zeitdiagramme (siehe dazu Bild 2.82)

Tafel 2.47
Zustandsfolgetabelle für den binären Vorwärtszähler nach Bild 2.99

Taktimpuls-zahl	a_4	a_3	a_2	a_1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Der asynchrone Binärzähler hat zwei Nachteile:

1. Die Verzögerungszeiten aller vorhergehenden Trigger und die Verzögerungszeit der betrachteten Triggerstufe addieren sich, so daß die Verzögerung mit der Stufenzahl zunimmt.
2. Bei der Decodierung der Zählerzustände durch an die Ausgänge geschaltete Gatter kann es zur Spike-Bildung kommen (spike ≙ kurzer Zusatzimpuls).

Die Verzögerungszeit t_v **(Bild 2.101)** wirkt sich besonders dann aus, wenn man in der Nähe der oberen Grenzfrequenzen, also bei sehr kurzen Impulsabständen, arbeitet. Durch diese Verzögerungszeit kann es keinen Synchronismus zwischen den zum gleichen Umschaltpunkt gehörenden Schaltflanken geben. Daher auch die Bezeichnung *asynchroner* Zähler.
Zu einer Spike-Bildung kommt es z. B. bei dem Decoder, der auf $y = \bar{a}_1\bar{a}_2$ reagiert. Ohne Einfluß von Verzögerungszeiten müßte zur Zeit t_x (Bild 2.101) die Variable $y = \bar{a}_1\bar{a}_2 = 0$ sein, da a_2 zu diesem Zeitpunkt (ohne Verzögerungszeit) im Zustand 1 sein müßte. Das ist aber nicht der Fall, so daß hier während der Zeit t_x *die Variable* $y = 1$ wird. Diese Spike-Decodierung kann durch einen zusätzlichen Gattereingang vermieden werden, der durch einen zeitlich entsprechend verschobenen Impuls (Strobe-Impuls) getaktet wird.

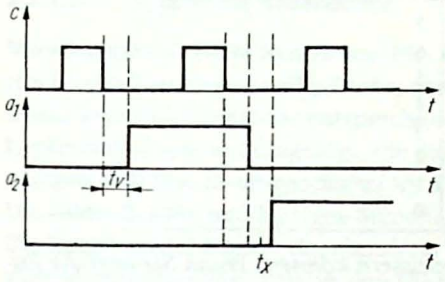
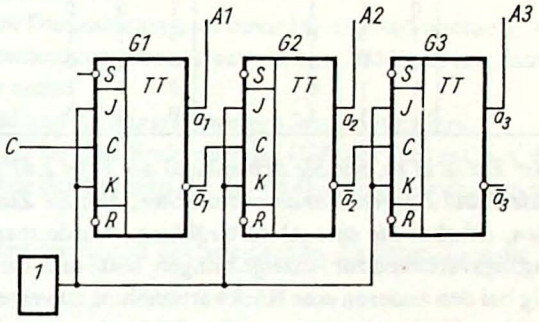


Bild 2.101. Darstellung der Verzögerungszeiten des asynchronen Binärzählers



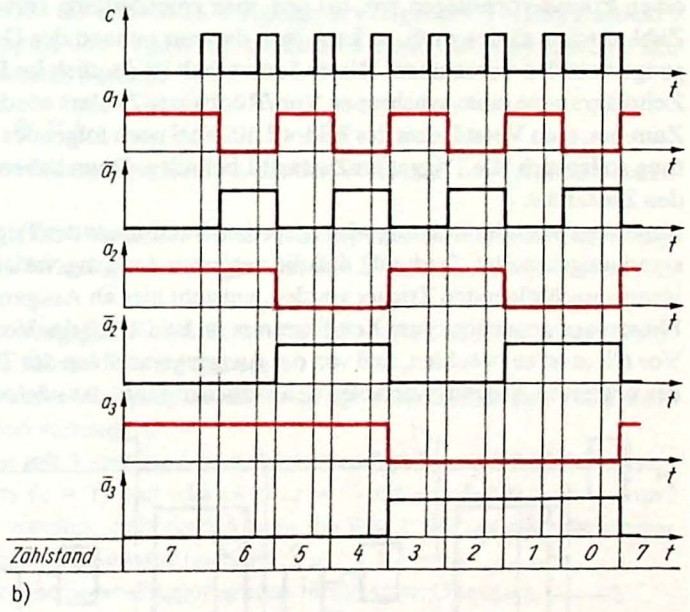


Bild 2.102
Asynchroner Rückwärtszähler aus JK-Master-Slave-Triggern
a) Signalflußbild; b) Zeitdiagramme

Rückwärtszähler. Einen asynchronen Rückwärtszähler erhält man aus JK-Master-Slave-Triggern, wenn man die negierten Ausgangsvariablen jeder Triggerstufe auf den Takteingang des jeweils nachfolgenden Triggers schaltet **(Bild 2.102, Tafel 2.48)**.
Im Prinzip verhält sich diese Schaltung ähnlich wie der Vorwärtszähler. Da jedoch die negierten Ausgangsvariablen die Umschaltung des jeweils nachfolgenden Triggers vornehmen, entsteht hier-

bei eine Bitmusterfolge der negierten Ausgangsvariablen, die der Vorwärtszählung entspricht, während die Bitmusterfolge der nichtnegierten Ausgangsvariablen eine Rückwärtszählung ergibt. *Anhand der Tafel 2.48 können Sie sich von diesem Sachverhalt überzeugen.*
Für diejenigen, die kritisch mitdenken, muß jetzt eigentlich folgende Frage entstehen: Ist dieser Sachverhalt nicht auch beim Vorwärtszähler nach Bild 2.99 gegeben, daß die negierten Ausgangsvariablen in der entgegengesetzten Richtung wie die nichtnegierten Ausgangsvariablen zählen?

Tafel 2.48. Zustandsfolgetabelle für den binären Rückwärtszähler nach Bild 2.102

Taktimpuls-zahl	Negierte Ausgänge			Direkte Ausgänge			Dezimalzahl des Bitmusters an den direkten Ausgängen
	$\bar{a}_3$	$\bar{a}_2$	$\bar{a}_1$	$\bar{a}_3$	$\bar{a}_2$	$\bar{a}_1$	
0	0	0	0	1	1	1	7
1	0	0	1	1	1	0	6
2	0	1	0	1	0	1	5
3	0	1	1	1	0	0	4
4	1	0	0	0	1	1	3
5	1	0	1	0	1	0	2
6	1	1	0	0	0	1	1
7	1	1	1	0	0	0	0

In der Tat ist es so, wie Sie sich anhand der Tafel 2.47 vergewissern können. Wenn Sie dort die Zustände 0 und 1 gegeneinander vertauschen, also die Zustände der negierten Ausgangsvariablen darstellen, erhalten Sie eine Abwärtszählung. Würde man also die negierten und die nichtnegierten Ausgangsvariablen zur Anzeige bringen, hätte man bei den einen eine Vorwärtszählung und gleichzeitig bei den anderen eine Rückwärtszählung zu verzeichnen.
Trotzdem ist der eben vorgestellte Zähler der eigentliche Rückwärtszähler. Wenn man nämlich einen Prozeß vorzuliegen hat, bei dem man eine Zeitlang vorwärts und dann von der erreichten Zahl abwärts zählen muß, so kann man dies nur anhand der Darstellung ein und derselben Ausgangsvariablen vornehmen. Dieser Sachverhalt ist deutlich im Bild 2.105b dargestellt, in dem die Zeitdiagramme eines synchronen Vor-/Rückwärts-Zählers wiedergegeben sind.
Zum besseren Verständnis des Bildes 2.102b sei noch folgendes erläutert: Zu Beginn der Betrachtung sollen sich alle Trigger im Zustand 1 befinden. Dann haben alle negierten Ausgangsvariablen den Zustand 0.
Nach dem Start der Zählung wird mit jedem Taktimpuls der Trigger G1 jeweils in den anderen Zustand umgeschaltet. Dadurch, daß die negierten Ausgangsvariablen zu den Eingangsvariablen der jeweils nachfolgenden Trigger werden, entsteht hier ab Ausgang des Triggers G1 ein Wechsel der Phasenlage gegenüber dem Zeitdiagramm im Bild 2.100 des Vorwärtszählers.
Vor allem ist zu beachten, daß von der Ausgangsvariablen des Triggers G1 ab jetzt die 1/0-Flanken der negierten Ausgangsvariablen den Umschaltpunkt der nächsten Triggerstufe bestimmen.

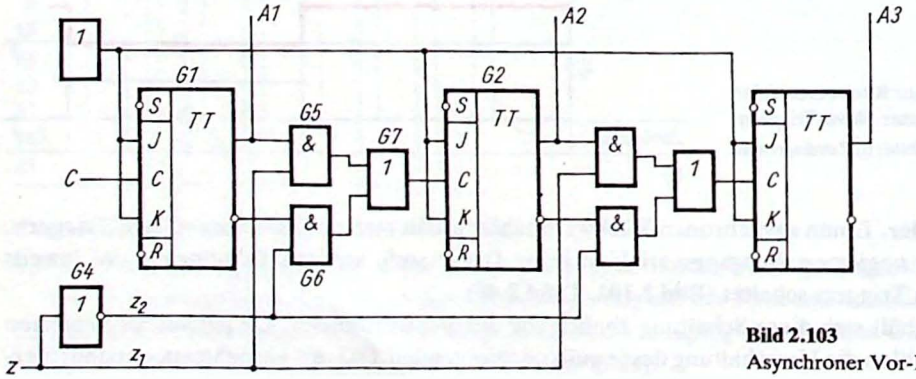


Bild 2.103
Asynchroner Vor-Rückwärts-Zähler

Asynchroner Vor-/Rückwärts-Zähler. Durch einen 2 : 1-Multiplexer (G5, G6, G7) nach **Bild 2.103** zwischen jeder Triggerstufe kann man den Zähler wahlweise in den Vorwärts- oder Rückwärtszählbetrieb schalten (programmieren).
Ist z_1 im Zustand $z_1 = 1$, so zählt der Zähler vorwärts; ist $z_2 = 1$, so zählt er rückwärts.
Um zu vermeiden, daß beide z-Werte gleichzeitig den Zustand 1 annehmen, kann man einen Inverter G4 einschalten. Mit ihm gilt

$z = 0$ Rückwärtszählen

$z = 1$ Vorwärtszählen.

2.2.4.2. Synchrone Binärzähler

Vorwärtszähler. Beim Synchronzähler wird allen Zählstufen der Taktimpuls zugeführt, so daß ein *synchrones* Umschalten aller Stufen gewährleistet wird.
Damit aber die Stufen nur entsprechend dem Dualzahlensystem ihren 1/0-Zustand wechseln, sind Logikschaltungen mit eingefügt, die die Entscheidung übernehmen, ob eine Zählstufe mit dem jeweiligen Taktimpuls umgeschaltet wird oder nicht.
Die **Bilder 2.104 a und b** zeigen Signalflußbild und Zeitdiagramm eines Synchronzählers.
Die Eingänge der Master der einzelnen Trigger werden direkt oder mit Hilfe der Gatter G5, G6 jeweils so voreingestellt, daß bei den 1/0-Flanken der Taktimpulse das gleichzeitige Umschalten aller in Frage kommenden Slaves gesichert ist.
Sehen wir uns den Zustand kurz vor Eintreffen des vierten Taktimpulses an. Es liegen dann die Zustände vor:

$$a_1 = 1, \quad a_2 = 1, \quad a_3 = 0.$$

Damit sind dann über das UND-Glied G5 und die JK-Eingänge des Triggers G3 in den Zustand 1 gesetzt, so daß der nächste Taktimpuls den Trigger G3 umschalten kann. Mit dem vierten Taktimpuls sollen folgende Zustände erreicht werden:

$$a_1 = 0, \quad a_2 = 0, \quad a_3 = 1,$$

wobei der Zustandswechsel aller drei Ausgangsvariablen mit der 1/0-Flanke des Taktimpulses erfolgen soll.
Dazu werden alle drei Master mit der 0/1-Flanke des vierten Taktimpulses bereits in die zugehörige Lage gebracht, so daß sich mit der 1/0-Flanke von c die gleichzeitige Übernahme in die Slaves vollziehen kann.
Noch deutlicher ist dieser Vorbereitungsprozeß bei Eintreffen der 0/1-Flanke des achten Taktimpulses zu erkennen. Hier werden alle vier Master vorbereitet.
Durch dieses gleichzeitige Umschalten aller betreffenden Ausgänge wird die Bildung von Spikes (kurze Zusatzimpulse) weitestgehend verhindert.
Synchroner Vor-/Rückwärts-Zähler mit Voreinstell- und Rücksetzmöglichkeit. Bild 2.105 a zeigt einen Synchronzähler, der vorwärts ($z = 1$) und rückwärts ($z = 0$) zählen, der rückgesetzt und über die Dateneingänge D1 bis D3 voreingestellt werden kann. Im Bild 2.107 b ist ein zugehöriges Zeitdiagramm dargestellt, das folgende Phasen demonstriert:
Phase A: Rücksetzen. Alle Master- und Slave-Trigger werden rückgesetzt. Damit ist
$a_1' = a_1 = a_2' = a_2 = a_3' = a_3 = 0$.
Phase B: Die 0/1-Flanke des ersten Taktimpulses bringt a_1' in den Zustand 1.
Phase B': Die 1/0-Flanke des ersten Taktimpulses bewirkt: $a_1 = 1, \bar{a}_1 = 0$ (Zählzustand 1).
Phase C: Die 0/1-Flanke des zweiten Transistors bewirkt: $a_1' = 0, a_2' = 1$.
Phase C': Die 1/0-Flanke des zweiten Taktimpulses zieht nach sich:
$a_1 = 0, \quad \bar{a}_1 = 1, \quad a_2 = 1, \quad \bar{a}_2 = 0$ (Zählzustand 2).

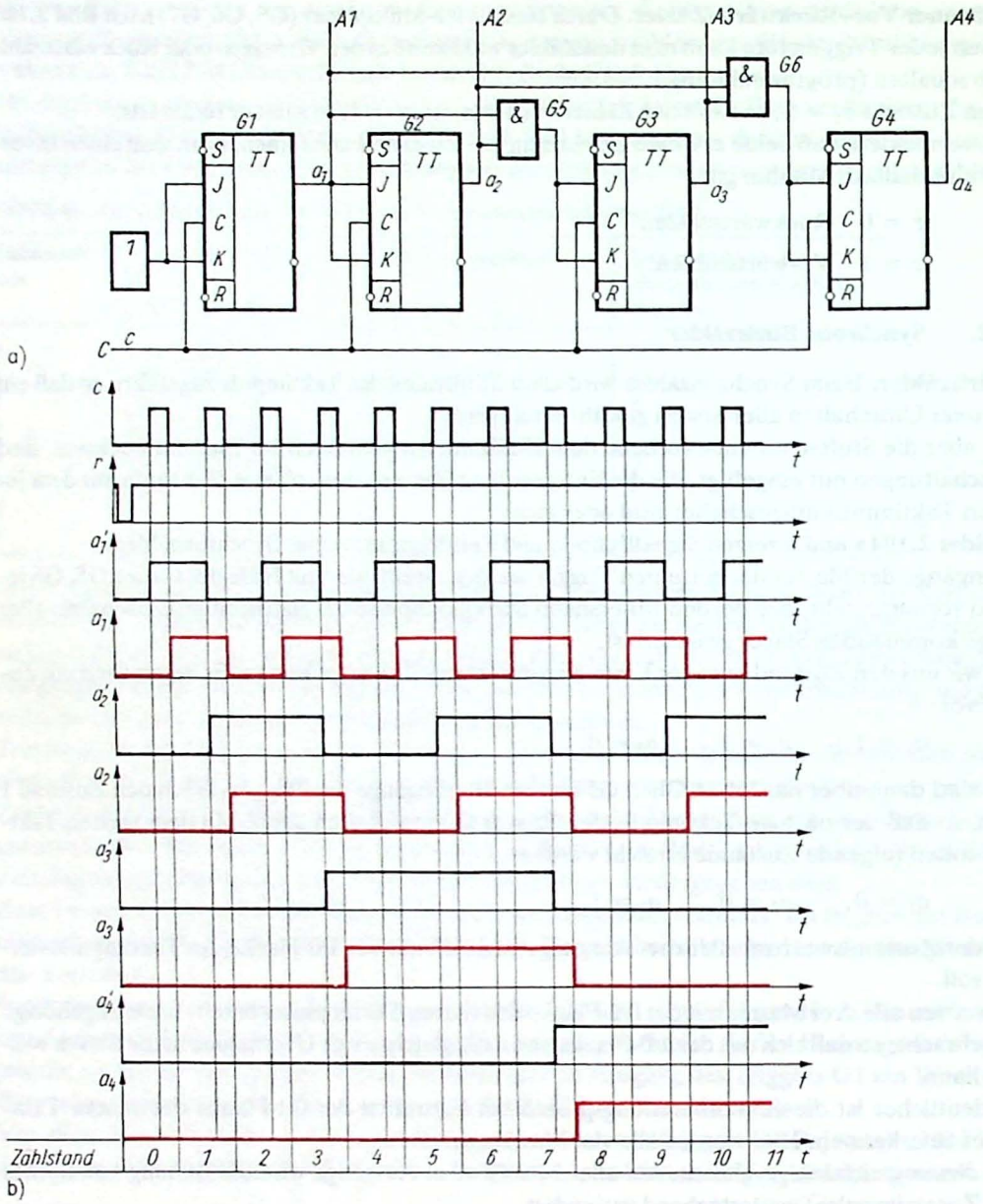
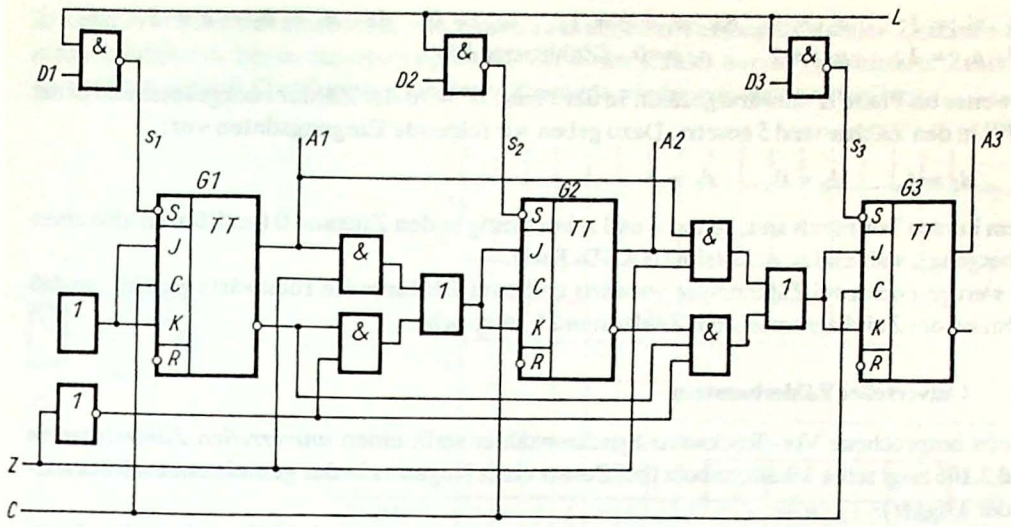
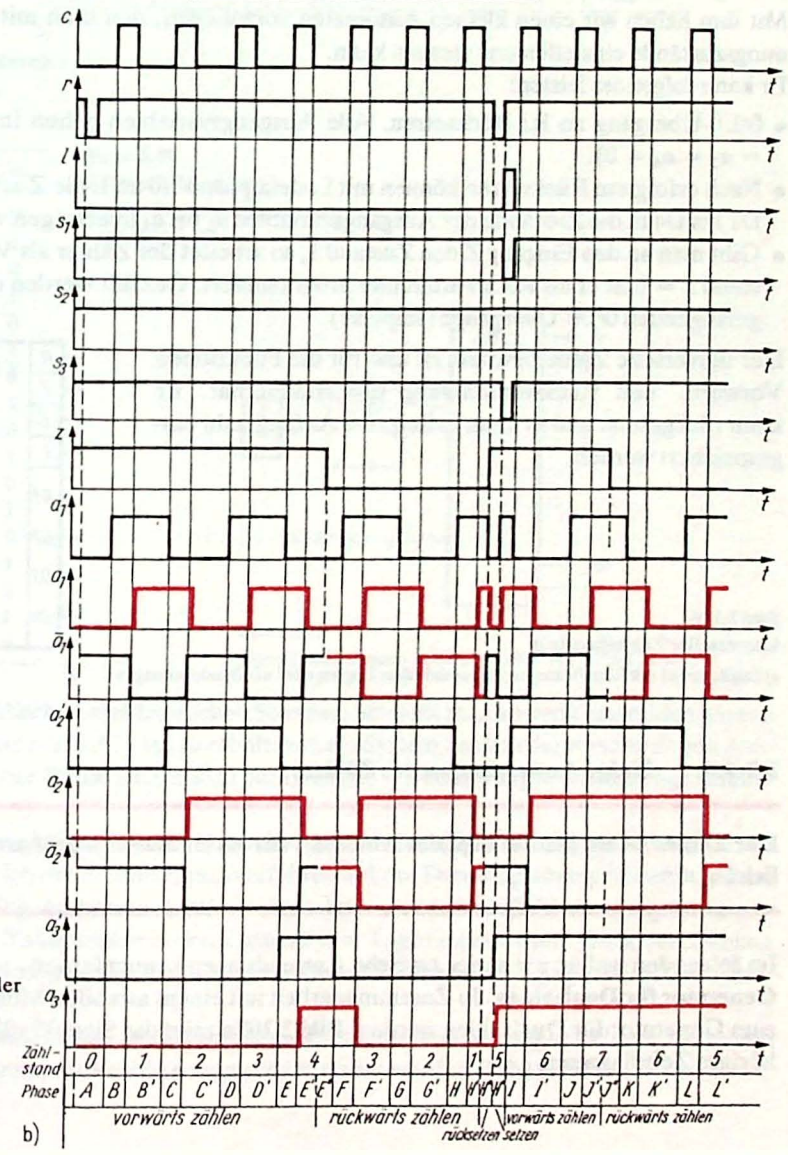


Bild 2.104. Synchroner Binärzähler aus JK-Master-Slave-Triggern
a) Signalflußbild; b) Zeitdiagramme

So wird weiter vorwärts gezählt, bis der Zählzustand 4 erreicht ist. In der Phase E'' geht z in den Zustand $z = 0$, so daß von hier ab rückwärts gezählt wird. Durch diese Umschaltung von z ändert sich beim Trigger G1 nichts. Die Trigger G2 und G3 werden aber nun an ihren JK-Eingängen von den negierten Ausgangsvariablen $\bar{a}_1$ und $\bar{a}_2$ der vorgegebenen Zählerbausteine beeinflußt. Daher folgt, daß der Master von G2 nur dann umgeschaltet wird, wenn $\bar{a}_1$ und c den Zustand 1 aufweisen. Der Master von G3 wird umgeschaltet, wenn $\bar{a}_1$ und $\bar{a}_2$ und c im Zustand 1 sind.
Unter Beachtung dieser Zusammenhänge kann man dann den weiteren Verlauf des Zeitdiagramms ab Phase F verstehen.



a)



b)

Bild 2.105
Synchroner Vor-Rückwärts-Zähler mit Voreinstell- und Rücksetzmöglichkeit
a) Signalflußbild; b) Zeitdiagramme

Phase F: $a_1' := 1$, $a_2' := 1$, da $\bar{a}_1 = c = 1$, $a_3' := 0$, da $\bar{a}_1 = \bar{a}_2 = c = 1$.
Phase F': $a_1 := 1$, $a_2 := 1$, $a_3 := 0$ (Zählzustand 3).

So wird weiter bis Phase H' abwärts gezählt. In der Phase H'' wird der Zähler rückgesetzt und in der Phase H''' in den Zählzustand 5 gesetzt. Dazu geben wir folgende Eingangsdaten vor:

$$d_1 = 1, \quad d_2 = 0, \quad d_3 = 1 .$$

Mit einem kurzen Torimpuls an L gehen s_1 und s_3 kurzzeitig in den Zustand 0 (vollführen also einen 1/0/1-Übergang), während $s_2 = 1$ bleibt (NAND-Tor!).
Danach werden noch zwei Zählimpulse vorwärts und zwei Zählimpulse rückwärts gezählt, so daß der Endstand des Zeitdiagramms dem Zählzustand 5 entspricht.

2.2.4.3. Universeller Zählerbaustein

Der zuletzt besprochene Vor-/Rückwärts-Synchronzähler stellt einen *universellen Zählerbaustein* dar. **Bild 2.106** zeigt seine Schaltsymbole (bei Zusatz eines Negators in der gemeinsamen Rücksetzleitung der Trigger).
Mit ihm haben wir einen kleinen Automaten vorzuliegen, den man mit Hilfe elektrischer Spannungszustände einstellen und steuern kann.
Er kann folgendes leisten:

- 0/1/0-Übergang an R: Rücksetzen. Alle Ausgangsvariablen gehen in den Zustand 0 ($a_1 = a_2 = a_3 = a_4 = 0$).
- Nach erfolgtem Rücksetzen können mit Ladeimpuls 0/1/0 an L die Zustände der Dateneingänge D1 bis D4 in die Zustände der Ausgangsvariablen a_1 bis a_4 übertragen werden.
- Gibt man an den Eingang Z den Zustand 1, so arbeitet der Zähler als Vorwärtszähler. Beim Zustand $z = 0$ ist er als Rückwärtszähler programmiert. Gezählt werden die an den Takteingang C gelangenden 0/1/0-Übergänge (Impulse).

Der universelle Zählerbaustein ist also für die Funktionen Vorwärts- und Rückwärtszählung programmierbar. Er kann rückgesetzt und es kann in ihm eine Anfangszahl eingespeichert werden.

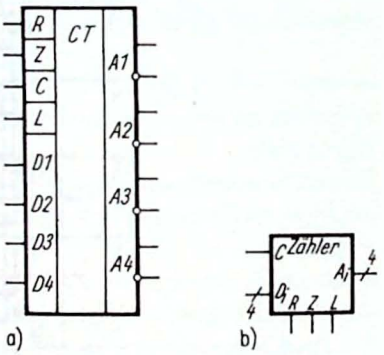


Bild 2.106
Universeller Zählerbaustein
a) Logiksymbol mit Einzelleitungen; b) vereinfachtes Logiksymbol mit Bündelleitungen

2.2.4.4. Einige Anwendungen des Zählers

Der Zähler ist ein kleiner digitaler Automat, der als Baustein in größeren Automaten unentbehrlich ist.

Im folgenden wollen wir einige typische Anwendungen kennenlernen.
Generator für Dualzahlen. In Zusammenarbeit mit einem astabilen Multivibrator kann ein Zähler zum Generator für Dualzahlen werden. **Bild 2.107 a** zeigt das Signalflußbild, Bild 2.107 b ein zugehöriges Zeitdiagramm.

In **Tafel 2.49** sind die einzelnen, von einem zum anderen Taktimpuls stabilen Zustände der Ausgangsvariablen a_1 bis a_4 zusammengestellt. Da dieser Zähler nur 16 verschiedene Zustände aufweist, fängt er nach Durchlaufen all dieser Zustände wieder von vorn an zu zählen.

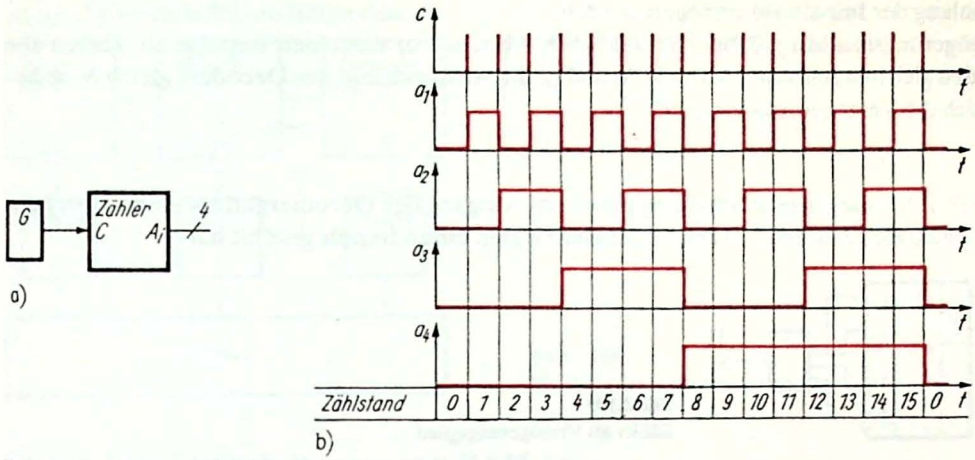


Bild 2.107. Generator für Dualzahlen
a) Signalflußbild; b) Zeitdiagramme

Impulszahl	a_4	a_3	a_2	a_1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Tafel 2.49
Zustandsfolgetabelle
für den Dualzahlengenerator
nach Bild 2.107

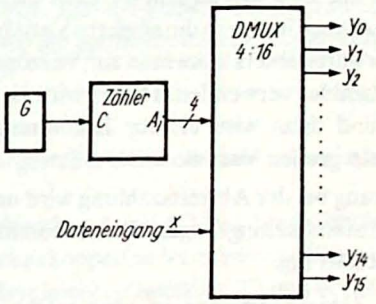


Bild 2.108
Signalflußbild eines zyklischen elektronischen Schalters

Zähler als Element für zyklischen elektronischen Schalter. Schließt man an den Dualzahlengenerator noch einen Demultiplexer (s. S. 61) an, so erhält man ein System, in dem die verschiedenen Ausgänge des Demultiplexers der Reihe nach und in periodischer Wiederholung in den durchgeschalteten Zustand kommen.

Damit haben wir dann einen *zyklischen* elektronischen Umschalter **(Bild 2.108)** geschaffen, wie wir ihn für das auf S. 60 beschriebene Zeitmultiplexverfahren auf der Demultiplexerseite benötigen.

Legt man auf den Dateneingang konstant auf $x = 1$, so bedeutet dies, daß die Ausgangsleitungen der Reihe nach für je eine Taktperiode in den Zustand $y_i = 1$ gebracht werden. Diese Anordnung übt dann die gleiche Funktion aus, wie diejenige, die wir auf S. 73 zur Steuerung eines ROM benötigt haben.

Zähler als Verzögerungseinrichtung. Ein Zähler kann in Zusammenschaltung mit einem Taktgenerator und einem Decoder auch als *Verzögerungsglied* wirksam werden **(Bild 2.109)**.

Dazu wird der Zähler anfangs rückgesetzt und bekommt dann Zählimpulse zugeführt. Der Decoder ist so beschaffen, daß er bei Auftreten einer gewählten Dualzahl ein Ausgangssignal abgibt. Damit kann dann ein bestimmter Vorgang vom Moment des Rücksetzens des Zählers bis zur erfolgten Abzählung der Impulszahl verzögert werden.
Die Verzögerungszeit hängt dabei vom zeitlichen Abstand der einzelnen Impulse ab. Haben alle Impulse den gleichen zeitlichen Abstand Δt und ist die Ansprechzahl des Decoders gleich N, so berechnet sich die Verzögerungszeit t_v zu

$$t_v = N \cdot \Delta t.$$

Die im Bild 2.109 dargestellte Schaltung liefert am Ausgang des Decodiergatters einen Übergang vom Zustand 0 zum Zustand 1, wenn der Zähler bis zum elften Impuls gezählt hat.

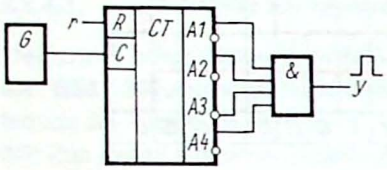


Bild 2.109
Zähler als Verzögerungsglied

Die dezimale 11 entspricht der Dualzahl

1011
$a_1 = 1$
$a_2 = 1$
$a_3 = 0$
$a_4 = 1.$

Folglich müssen zur Decodierung dieser Zahl die Eingänge des UND-Gliedes mit den direkten Ausgängen A1, A2, A4 und dem die negierte Variable liefernden A3-Ausgang verbunden werden.
Anstelle des Vorwärtszählers kann man zur Verzögerung um eine bestimmte Zeit $t_v = N \cdot \Delta t$ auch einen Rückwärtszähler verwenden. Dieser wird zu Beginn auf die Zahl der Verzögerungsschritte voreingestellt, und dann wird bis zur Erkennung des Nullzustands rückwärts gezählt. Diese Variante hat einen großen Vorteil:

- Zur Decodierung bei der Abwärtszählung wird unabhängig von der voreingestellten Zahl immer nur der Nullzustand herangezogen. Damit kommt man für alle Zahlen mit einem einzigen, sehr einfachen Decoder aus.

In integrierten Zählerbausteinen ist eine solche Decodierschaltung zur Erkennung des Nullzustands enthalten.
So weisen z. B. die universellen Zählerbausteine der TTL-Serie (D 192/SN 74 192 N und D 193/SN 74 193 N) jeweils zwei Übertragsausgänge $\ddot{u}_v$ und $\ddot{u}_r$ auf. Am Übertragsausgang $\ddot{u}_v$ wird ein Übertragsimpuls abgegeben, wenn der Zähler beim Vorwärtszählen von der höchsten Zahl zur Zahl Null umgeschaltet wird. Am Ausgang $\ddot{u}_r$ entsteht dieser Impuls dann, wenn der Zähler beim Rückwärtszählen von Null zur höchsten Zahl umgeschaltet wird.
Zähler zur Uhrzeitbildung. Eine hochgenaue Uhr erhält man durch Ableiten eines Sekundenimpulstakts aus einer quarzstabilisierten Schwingung.
Zählt man die Sekundenimpulse, so ergeben 60 Sekundenimpulse einen Minutenimpuls, 60 Minutenimpulse einen Stundenimpuls. Nach $60 \times 60 \times 24$ Sekundenimpulsen muß der Gesamtzähler wieder seine Nullstellung erreichen. **Bild 2.110** zeigt das Blockbild einer solchen elektronischen Uhr.
Der im Bild 2.110 als Block dargestellte Frequenzteiler besteht ebenfalls aus Zählbausteinen, die so geschaltet sind, daß sie nach einer bestimmten Impulszahl wieder auf Null zurückgestellt wer-

den. So teilt z. B. ein Zähler, der bis zehn zählt und dann wieder bei null von vorn beginnt und bei jedem zehnten Eingangsimpuls einen Ausgangsimpuls abgibt, die Zahl der Eingangsimpulse genau durch zehn. Durch Rückkopplungsmaßnahmen lassen sich auch beliebige andere, ganzzahlige Teilerverhältnisse herstellen.

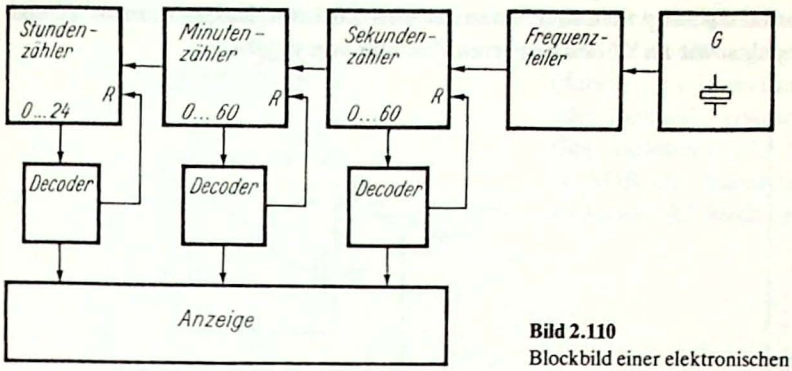


Bild 2.110
Blockbild einer elektronischen Uhr

2.2.5. Speicherschaltungen vom RAM-Typ

Der RAM (random access memory) ist ein Schreib-Lese-Speicher. Er hat den Vorteil, daß sein Speicherinhalt ohne großen Aufwand und Zeitverlust während des normalen Betriebs verändert werden kann.

Seine Speicherelemente sind wie beim ROM in Matrixform angeordnet und können aus *statisch* oder *dynamisch* wirkenden Baugruppen bestehen. Um möglichst viele Speicherzellen auf einen Halbleiterchip zu bekommen, hat man raumsparende Speicherelemente entwickelt.

2.2.5.1. Bipolare Speicherzelle

Für bipolare RAM hat man die im **Bild 2.111** dargestellte Zellenstruktur entwickelt, die wie ein getakteter RS-Trigger wirkt.
Die bistabile Schaltung, bestehend aus den Widerständen R1, R2 und den 2-Emitter-Transistoren T1, T2, die man auch als zweistufigen, stark rückgekoppelten Verstärker auffassen kann, ist völlig symmetrisch aufgebaut. Das bedeutet, T1 hat die gleichen Daten wie T2 und R1 ist gleich R2.
Trotz der Symmetrie sind die Potentiale an den Punkten A1 und A2 im statischen Zustand nie gleich. Der Grund für dieses Verhalten ist die Rückkopplung zwischen den beiden Verstärkerstufen und die Verstärkung der Stufen im Aussteuerbereich.
Nehmen wir einmal an, daß an A1 ein schwach positives Potential liege, das den Transistor T2 sperre. Dann wird das Potential an A2 $\varphi_{A2} = +5$ V. Durch die direkte Verkopplung der Basis B1 mit A2 gelangen diese 5 V auch auf die Basis des Transistors 1, wodurch dieser voll durchgesteuert wird, so daß sich tatsächlich das ausgangs angenommene schwach positive Potential an A1 einstellt. Hätte man das kleinere positive Potential an A2 angenommen, so wäre T2 als aufgesteuert und T1 als gesperrt ermittelt worden.
Die Schaltung zeigt also zwei stabile Zustände, in denen sie verharren kann:

1. T1 geöffnet, T2 gesperrt (Speicherzustand 0)
2. T1 gesperrt, T2 geöffnet (Speicherzustand 1).

Der ersten stabilen Lage wollen wir den Informationszustand 0, der zweiten Lage den Zustand 1 zuordnen.

Von dem einen in den anderen stabilen Zustand kann man die Schaltung durch entsprechende Impulsgabe bringen:

- Vom ersten Zustand gelangt man zum zweiten stabilen Zustand, indem man an Y1 und an ADR einen positiven Impuls sowie an Y2 den negierten Zustand von y_1 gibt.
- Vom zweiten stabilen Zustand gelangt man zum ersten stabilen Zustand, indem man an Y2 und ADR einen positiven Impuls sowie an Y1 den negierten Zustand von y_2 gibt.

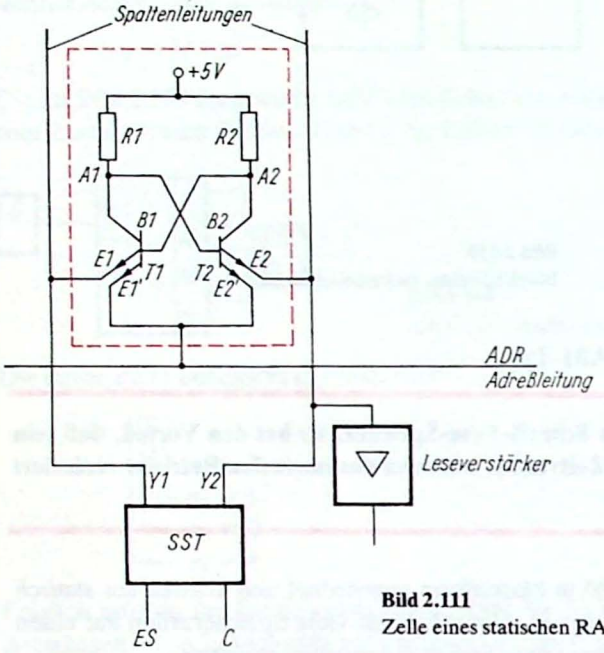


Bild 2.111
Zelle eines statischen RAM in Bipolartechnik

Beweisen wir die letzte Aussage. Der zweite stabile Zustand bietet folgende Ausgangssituation: T1 gesperrt, T2 geöffnet. In diesem Zustand fließt der Kollektorstrom von T2 im wesentlichen über E2′ und die Adreßleitung zurück. Das wird dadurch erreicht, daß die Potential- und Widerstandsverhältnisse an beiden Emittern entsprechend unterschiedlich gemacht werden.
Gibt man nun zunächst den positiven Impuls (+5 V) an ADR, so wird der Emitter E2′ gesperrt, und der Kollektorstrom muß über E2 abfließen. Kommt nun noch der positive Impuls an Y2 hinzu, der durch die Schreibsteuerung (SST) gebildet wird, so wird der Transistor T2 völlig gesperrt. Das führt zu $\varphi_{A2} = +5$ V und damit zur Öffnung von T1. Der nun durch T1 fließende Kollektorstrom kann zunächst nur über E1 zurückfließen. Nach Verschwinden der positsiven Impulse an Y2 und ADR bleibt die bistabile Speicherzelle in ihrer neuen Lage. Damit hat sie nun den Informationszustand 0 gespeichert. Der Strom durch T1 fließt dann vorwiegend über E1′ und die Adreßleitung.
Zum Auslesen des Speicherinhalts gibt man nur einen positiven Impuls an ADR. Ist dabei z. B. T2 geöffnet (Speicherzustand 1), so bewirkt diese Impulsgabe, daß der bisher im wesentlichen über E2′ fließende Kollektorstrom von T2 nun über E2 fließen muß. Dabei entsteht in der rechten Spaltenleitung ein positiver Impuls, der vom Leseverstärker übernommen wird. Ist T2 gesperrt (Speicherzustand 0), gibt es einen positiven Impuls in der linken Spaltenleitung, der aber nicht den Leseverstärker beeinflußt.

2.2.5.2. MOS-Speicherzelle

Mit MOS-Transistoren kann der Leistungsbedarf eines Speichers drastisch gesenkt und damit eine höhere Packungsdichte als bei bipolaren Transistoren erreicht werden. Allerdings sind MOS-Speicher-Anordnungen langsamer.

Statische MOS-Speicherzelle. Der Aufbau der im **Bild 2.112** dargestellten Speicherzelle ist dem der bipolaren Speicherzelle ähnlich. Die Transistoren T3 und T4 wirken als Arbeitswiderstände der Transistoren T1 und T2. Auch bei dieser Rückkopplungsschaltung gibt es nur zwei stabile Zustände, in denen jeweils einer der beiden Transistoren T1, T2 geöffnet und der andere gesperrt ist.
Mit Hilfe eines Impulses auf ADR werden die beiden Transistoren T5 und T6 aufgesteuert und die an E1 und E2 liegenden Informationszustände (dabei $e_1 = \bar{e}_2$) zum Einstellen des benötigten stabilen Zustands verwendet.
Das Auslesen erfolgt wieder durch Impulsgabe an ADR und Auswertung der Potentialzustände an A1 und A2 durch Leseverstärker.

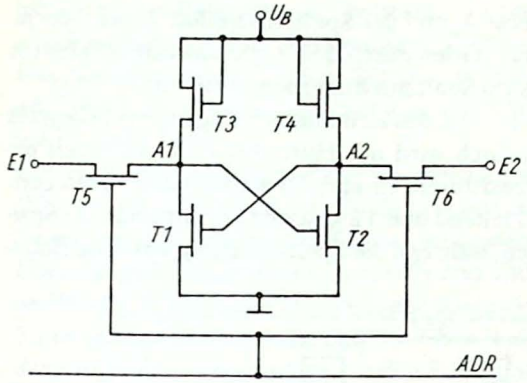


Bild 2.112
Zelle eines statischen RAM in MOS-Technik

Dynamische MOS-Speicherzelle. Nach dem gleichen Prinzip wie schon auf S. 110 beim dynamischen Schieberegister, kann man auch dynamische MOS-Speicherzellen aufbauen. **Bild 2.113** zeigt eine dynamische Speicherzelle mit drei Transistoren.
Zum Einschreiben wählt man die *Spaltenleitung* durch negatives Potential und die Zeilenauswahlleitung *Schreiben* (ebenfalls mit negativem Potential) an. Dadurch wird T1 geöffnet, und C kann geladen werden. Auch hier besteht C nur aus den Schalt- und Transistorkapazitäten. Diese Ladung bleibt für einige Millisekunden bestehen. Um den gespeicherten Zustand längere Zeit aufrechtzuerhalten, müssen die Ladungszustände aller Zellen eines solchen RAM im Abstand von einigen Millisekunden regeneriert werden. Dazu muß der bisherige Zustand gelesen und bei Feststellung des eingeschriebenen Zustands C frisch geladen werden.

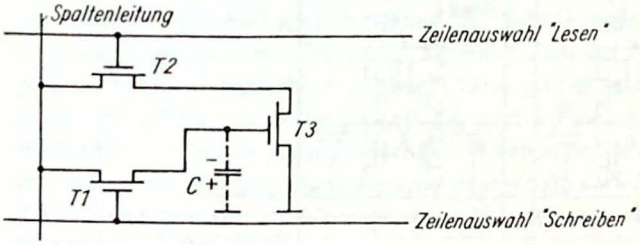


Bild 2.113. Zelle eines dynamischen RAM mit drei Transistoren in MOS-Technik

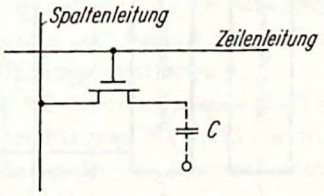


Bild 2.114. Zelle eines dynamischen RAM mit einem Transistor in MOS-Technik

Zum Lesen wird neben der Spaltenleitung die Zeilenauswahlleitung *Lesen* an negatives Potential geschaltet. War C geladen, so erfolgt Stromfluß durch T2 und T3. Der damit verbundene Spannungsimpuls an der Spaltenleitung wird in einem (nicht dargestellten) Leseverstärker ausgewertet.
Die dynamische 3-Transistor-Speicherzelle ist platzsparend und garantiert dadurch höhere Pakkungsdichten.
Noch mehr Platz wird gespart, wenn die dynamische 1-Transistor-Zelle angewendet wird **(Bild 2.114)**.
Allerdings muß hier eine wesentlich empfindlichere und kompliziertere Lese- und Auffrischlogik verwendet werden. Schreiben und Lesen dieser 1-Transistor-Zelle werden nur mit einer einzigen Zeilenleitung angefordert. Das hat jedoch den Nachteil, daß sich *C* beim Lesen entlädt, so daß dann ein nachträgliches Auffrischen der Ladung außer der Reihe erfolgen muß.

2.2.5.3. Aufbau von RAM

SRAM. Im **Bild 2.115** ist die Prinzipschaltung eines RAM mit statischen MOS-Speicherzellen (SRAM) dargestellt. Er dient zur Speicherung von $n \times m$ Bit, die über den Eingang D_i/D_0 bei $\overline{CS} = 0$ und $\overline{WE} = 0$ in die Speichermatrix (3) eingeschrieben und bei $\overline{CS} = 0$ und $\overline{WE} = 1$ gelesen werden.

Durch die entsprechende Belegung der Zeilenadreßbit A_i und der Spaltenadreßbit A_j wird die jeweils zu aktivierende Speicherzelle ausgewählt. Dazu werden durch den Zeilendecoder (2) jeweils eine Zeile und durch den Spaltendecoder (5) jeweils ein Spaltenleitungspaar aktiviert.

So vollzieht sich der Lesevorgang der Zelle 0,0 wie folgt: An den Zeilenadreßeingängen (1) liegt ein Adreßbitmuster, das durchweg Nullen enthält. Dadurch wird mit Hilfe des Zeilendecoders die Zeile 0 an aktives Potential geschaltet, und die Transistoren T5 und T6 werden durchgesteuert. Unter der Annahme, daß in der Speicherzelle 0,0 T1 leitend und T2 gesperrt ist, wird nun die Spaltenleitung Sp 01 auf das Logik-Potential Null entladen, während die Spaltenleitung Sp 02 auf Batte-

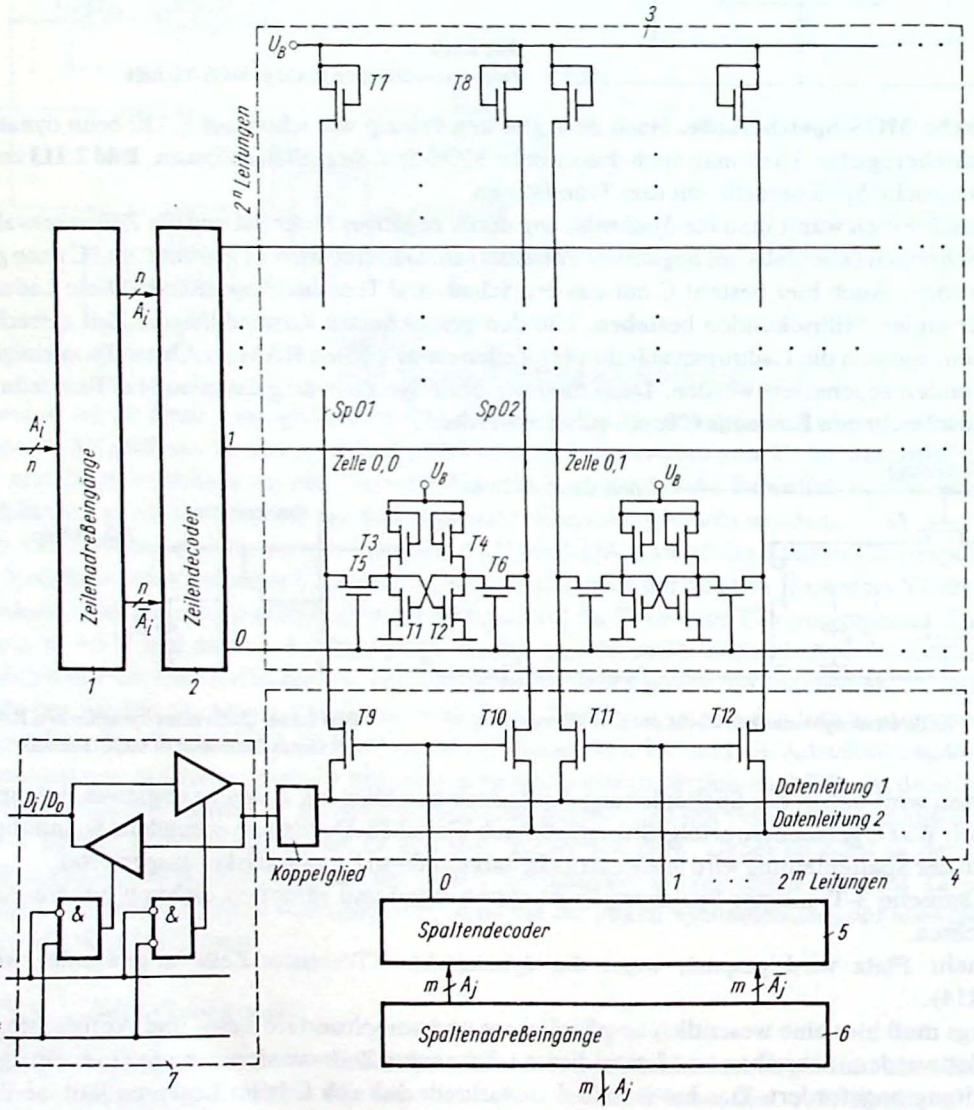


Bild 2.115. Prinzipschaltung eines RAM mit statischen MOS-Speicherzellen

riepotential geladen bleibt. (Über die als Widerstände geschalteten Transistoren T7 und T8 sind zuvor die beiden Spaltenleitungen, wie auch alle anderen Spaltenleitungen, auf U_B aufgeladen worden.)

An den Spaltenadreßeingängen (6) muß für unser Beispiel ebenfalls das Bitmuster 0 liegen, so daß der Spaltendecoder über seinen Ausgang 0 die Transistoren T9 und T10 durchsteuern kann. Damit wird dann das Potential der Spaltenleitung Sp 01 auf die Datenleitung 1 und das Potential der Spaltenleitung Sp 02 auf die Datenleitung 2 geschaltet.

Das Koppelglied reicht diese Signale, zu einem einzigen umgeformt, an die E/A-Stufe (7) weiter. Ist $\overline{CS} = 0$ und $\overline{WE} = 1$, so gelangt die Bitbelegung der Zelle 0,0 auf diese Weise zum kombinierten E/A-Pin D_i/D_0.

Das Einschreiben in die Zelle 0,0 vollzieht sich bei gleicher Adreßgabe sowie bei $\overline{CS} = 0$ und $\overline{WE} = 0$ so, daß die an Pin D_i/D_0 liegende Bitbewegung vom Koppelglied in ein leistungsstarkes Signal zur pegelrichtigen Belegung der beiden Datenleitungen 1 und 2 umgewandelt wird.

Über die geöffneten Transistoren T9 und T10 wird dann bei ebenfalls geöffneten Transistoren T5 und T6 der Trigger der Speicherzelle 0,0 in die gewünschte Lage gekippt.

Zu erwähnen bleibt noch, daß bei $\overline{CS} = 1$ der Pin D_i/D_0 hochohmig wird und daß der Spaltendecoder in Zusammenarbeit mit den Schalttransistoren T9, T10, T11, T12 und den gleichartigen anderen (4) als Spaltenmultiplexer wirkt.

Der im Bild 2.115 dargestellte RAM ist bitorientiert, da immer nur 1 Bit ein- bzw. ausgegeben werden kann. Es gibt aber auch RAM-Typen, die 4-Bit- oder 8-Bit-orientiert sind, also Halbbyte oder Byte ein-/ausgeben können. Dazu müssen die Tristate-Tore in der E/A-Stufe und die Datenleitungen in der Schreib-Lese-Schaltung (4) entsprechend mehrfach vorhanden sein.

DRAM. Bei dynamischen RAM (DRAM) kommen zur Prinzipschaltung nach Bild 2.115 noch so viele Auffrischverstärker und die entsprechenden Koppelglieder hinzu, daß beim Refresh-Vorgang alle Zellen einer Zeile auf einmal gelesen und mit ihrem Inhalt neu beschrieben werden können. Die Refresh-Steuerung kann auf dem Chip mitorganisiert werden. Bei einfacheren DRAM muß der Refresh jedoch von einer externen Steuereinrichtung (z.B. dem RAM-nutzenden Mikroprozessor) übernommen werden.

Bei einigen DRAM-Typen bestehen die Adreßeingänge aus Registern, in denen die Adresse zwischengespeichert werden kann. Dadurch wird der Adreßbus entlastet, da die Adreßbusbelegung nach dem Abspeichern wechseln kann. Adreßregister werden auch benutzt, um die Adresse in mehreren Schüben zu übertragen. Dadurch können Adreß-Pin eingespart werden.

NVRAM. Alle bisher aufgeführten RAM haben den Nachteil, daß sie ihren Speicherinhalt verlieren, wenn die Versorgungsspannung abgeschaltet wird. Daher hat man NVRAM (nonvolatile RAM = RAM mit nichtflüchtiger Speicherung) entwickelt, die aus der Zusammenschaltung von SRAM mit EEPROM bestehen.

Diese haben drei Betriebsarten:

1. Lesen des RAM (etwa 200 ns/Byte)
2. Schreiben des RAM (etwa 200 ns/Byte)
3. Umspeichern in den EEPROM (etwa 1 bis 10 ms/Byte).

Damit das Retten der Informationen vom SRAM in den EEPROM bei Stromausfall automatisch erfolgt, müssen Stromwächter ständig die Versorgungsspannung beobachten und im Notfall die notwendigen Maßnahmen einleiten. Solche Stromwächter werden bei weiterer Entwicklung auch mit auf dem IC des NVRAM aufgebracht sein, so daß Retten und Wiederübergabe an den SRAM vollautomatisch erfolgen können.

Infolge der zusätzlich notwendigen Bauelemente hat ein NVRAM nur 25 % der Speicherkapazität eines EEPROM mit der gleichen Chipfläche.

3. Informationsverarbeitung durch höher entwickelte Logiksysteme

Bisher haben wir eine Reihe von mehr oder weniger komplizierten Logiksystemen kennengelernt, die alle aus einfacheren Elementen zusammengesetzt sind.
So setzt sich z. B. ein Zähler aus einer Anordnung von Triggern, jeder Trigger aus einer Anordnung von elementaren Logikschaltungen zusammen.
Jede elementare Logikschaltung ist wiederum aus einzelnen Transistoren, Dioden, Widerständen aufgebaut.
Transistoren, Dioden, Widerstände werden in der Technik der integrierten Schaltungen in winzigen Abmessungen auf Substraten aus Halbleitermaterial durch verschiedene Dotierungen (pn-Übergänge) und mit Hilfe von Isolierzonen (z. B. SiO_2) und metallischen Elektroden realisiert.

Diese Untergliederung einer Schaltungsanordnung in einzelne Teilschaltungen, die wiederum aus noch einfacheren Teilen (Elementen) zusammengesetzt sind, läßt das Aufstellen einer Art Rangordnung zu, wie wir sie in **Tafel 3.1** finden.

Tafel 3.1. Rangstufen der Elemente von Logiksystemen

Rangstufe	Bezeichnung	Funktion
1	pn-Übergang	Element für Dioden, Transistoren, Widerstände
2	Diode	stromrichtungsabhängiger Widerstand (Gleichrichterwirkung)
3	Transistor	steuerungsfähiges Bauelement (Stromschalter)
4	elementare Logikschaltung	Realisierung logischer Elementarfunktionen
5	einfache Logikschaltung ohne Rückkopplung	Tor, Multiplexer, Demultiplexer, Codierer, Umcodierer, Decodierer, Vergleicher, ROM
6	einfache Logikschaltung mit Rückkopplung	Multivibratoren, Trigger, Register, Zähler
7	höher entwickelte Logiksysteme	Bitmustergenerator, spezielle Steuersysteme, universelle Steuersysteme

Bei Auswertung dieser Tafel könnte die Frage entstehen, ob es eine Erweiterung nach unten (unterhalb der Rangstufe 1) und nach oben (oberhalb des bisher Kennengelernten bis zur Rangstufe 6) gibt. Die Erweiterung nach unten führt uns zu den physikalischen Prozessen im Innern der Halbleiterkristalle. Diese werden im Wissenschaftsgebiet der Elektronenphysik erforscht und systematisiert. Die Erweiterung nach oben ist ebenfalls möglich und führt uns zu den höher entwickelten logischen Systemen. Wir wollen diese zunächst zusammenfassend in die Rangstufe 7 einordnen.
Die Schaltungen der Rangstufe 7 setzen sich aus Anordnungen aller vorhergehenden Rangstufen zusammen. Ihre Funktionen sind wiederum komplizierter und vielfältiger als diejenigen der vorhergehenden Stufen.
Bei der Behandlung der höher entwickelten Systeme werden wir ebenfalls den Weg von einfacheren zu komplizierteren Systemen gehen.

3.1. Bitmustergenerator

Als Bitmustergenerator wollen wir eine Logikschaltung bezeichnen, die laufend neue Bitbelegungen an parallelen Ausgängen erzeugt.

Bild 3.1 zeigt uns das grundlegende Signalflußbild, das aus Impulsgenerator, Zähler und ROM besteht. Diese Anordnung ist uns bereits bei der Besprechung des ROM-Einsatzes beim Waschmaschinenbeispiel begegnet, nur daß wir im Abschn. 2.1.7.7. noch nicht Generator und Zähler besprochen haben.

Da wir nach dem Studium der vorhergehenden Abschnitte alle notwendigen Kenntnisse besitzen, wollen wir die universelle Funktion dieses kleinen Automaten näher betrachten und noch einmal herausheben.

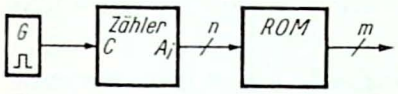


Bild 3.1
Signalflußbild eines Bitmustergenerators

Dieser Automat stellt die einfachste Form eines höher entwickelten Logiksystems dar und ist abgewandelt in allen diesen Systemen wiederzufinden. Der Taktgenerator als Schrittmacher liefert in gleichen Zeitabständen Impulse an den Zähler. Mit jedem Impuls bildet der Zähler ein neues paralleles Bitmuster, dessen aufeinanderfolgende Zustände der Folge der Dualzahlen entsprechen. Hat man z. B. einen Zähler aus drei Triggern vorzuliegen, so werden die Dualzahlen von 000 bis 111 (null bis sieben) gebildet. Danach wiederholt sich diese Folge von neuem. Die Ausgangszustände des Zählers werden als Adressen auf den ROM geschaltet und bewirken dadurch die nacheinander folgende Ausgabe der im ROM programmierten Bitmuster. Die ausgegebene Bitmusterfolge kann je nach Verabredung die unterschiedlichsten Steuerfunktionen ausüben.

Auch früher entwickelte mechanische oder elektromechanische Automaten enthielten einen solchen Bitmustergenerator. Das beste Beispiel gibt uns die Spieluhr, die nach folgendem Prinzip arbeitet: Auf einer sich drehenden Walze sind metallische Spitzen angebracht, die unter schwingungsfähigen Stahllamellen hinweggleiten, diese anheben und sie beim Verlassen zu Schwingungen anregen **(Bild 3.2)**. Auf diese Weise entstehen kurze Melodienstücke.

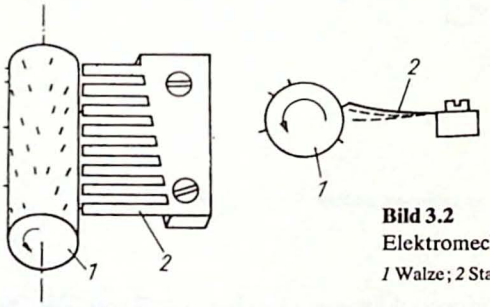


Bild 3.2
Elektromechanischer Bitmustergenerator für Spieluhren
1 Walze; *2* Stahllamellen

Auch die Drehorgel (Leierkasten) und das elektrische Klavier sind Beispiele für solche Automaten. Sie haben Bitmustergeneratoren. Die Walzenmuster sind Binärmuster und entscheiden je nach der Programmierung darüber, ob eine betreffende Tonschwingung erregt wird oder ausbleibt.

Mit welchem Teil der Spieluhr ist nun unser elektronischer Bitmustergenerator gleichwertig?

Mit der Walze, auf der die metallischen Spitzen sitzen!

Unser elektronisches System liefert in zeitlicher Reihenfolge unterschiedliche Spannungszustände an die Ausgangsklemmen. Die Walze der Spieluhr führt in zeitlicher Reihenfolge unterschiedliche

Stiftanordnungen unter die Stahllamellen. Sie kann durch unterschiedliche Anbringung der Stifte mit verschiedenen Mustern versehen werden. Beim elektronischen Bitmustergenerator kann der ROM für eine gewünschte Bitmusterfolge programmiert werden

Ein Vorteil des elektronischen Bitmustergenerators besteht darin, daß er keinerlei mechanisch bewegliche Teile besitzt. Daß er trotzdem seine Bitmuster zyklisch wiederholen kann, liegt daran, daß der vom Taktimpulsgenerator gespeiste Zähler jeweils nach Durchlaufen aller Zählerstände, immer wieder von neuem beim Zählzustand null beginnend, die Zählung wiederholt.

Ein weiterer Vorteil ist seine relativ einfache Programmier- und Umprogrammierbarkeit, die durch die Anwendung eines EPROM ermöglicht wird. Noch interessanter wird es, wenn man anstelle des ROM einen RAM verwendet. Da er wie der ROM nach Anschalten der Adressen die in den angewählten Zellen gespeicherten Zustände ausgibt, erhält man die gleiche Wirkungsweise – nur mit einem kleinen Unterschied:

- Der RAM kann kurzfristig (während des Betriebs) über elektrische Steuerung mit einem anderen Speicherinhalt versehen werden. Damit ist seine Bitmusterfolge an ein Prozeßgeschehen schnell anpaßbar.

Sie können sich jetzt vielleicht schon selbst vorstellen, welche Möglichkeiten sich dadurch eröffnen. Theoretisch lassen sich damit unendlich viele lösbare Steuerungsprobleme ausdenken.

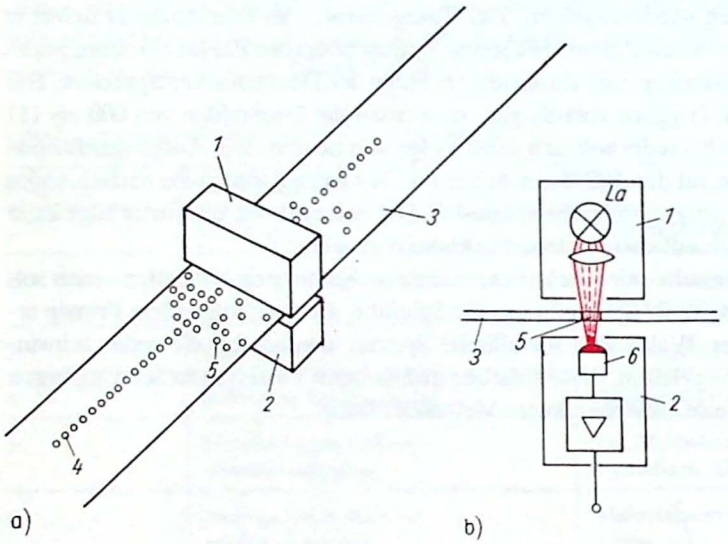


Bild 3.3. Prinzip des Lochstreifenlesers

a) Lochstreifen mit Abtastvorrichtung
1 Abtastvorrichtung (Lichtquelle); *2* Abtastvorrichtung (Lichtempfänger);
3 Lochstreifen; *4* Perforation zur Taktgabe und/oder Bandvorschub; *5* Löcher zur Informationsdarstellung

b) Abtastvorrichtung im Detail
6 Fotodiode

Im Prinzip liefert der Bitmustergenerator das gleiche Endprodukt wie ein Lochstreifenleser. Dieser liest zeittaktgesteuert die im Lochstreifen eingestanzten Lochkombinationen und bildet daraus elektrische Spannungszustände **(Bild 3.3)**. Während der Lochstreifen jedoch dem Verschleiß unterworfen und relativ materialaufwendig ist, wird beim elektronischen Speicher nur eine geringe elektrische Energie zur Speicherung oder Programmierung benötigt.

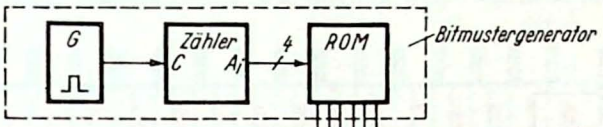
Schauen wir uns zum Abschluß der Betrachtung des Bitmustergenerators noch ein mögliches Anwendungsbeispiel an: die Steuerung einer wandernden Leuchtreklame **(Bild 3.4)**.

Um die Übersicht zu erleichtern, wollen wir ein einfaches Beispiel wählen und nicht gleich ganze Buchstaben und Zahlen, sondern nur einzelne Punkte wandern oder blinken lassen. Außerdem

wollen wir annehmen, daß der ROM bereits programmiert ist. Die im Bild 3.4 dargestellten roten Punkte, die das Leuchten der betreffenden Lampe andeuten sollen, entsprechen zugleich in ihrer Gesamtheit dem im ROM wirksamen Diodenmuster.
Für den Betrachter der sechs Lampen ergibt sich folgender Bewegungsablauf:

- Beim Zählzustand 0 sind alle Lampen dunkel.
- Vom Zählimpuls zwei bis sechs wandert ein Leuchtpunkt mit dem Zählimpulstakt von rechts nach links.
- Ab Zählimpuls sieben werden von links nach rechts der Reihe nach alle Lampen eingeschaltet, bis nach dem 12. Impuls alle Lampen leuchten.
- Von Impuls 13 bis 15 blinken die Lampen *La3* und *La4*.
- Danach beginnt der Vorgang von neuem.

Anstelle der Lampen in unserem Beispiel können natürlich beliebige andere Signal- und Zeichengeber betätigt werden. Es können auch Stelleinrichtungen technologischer Prozesse nach einem geplanten Ablauf beeinflußt werden, so daß ein straffes Zeitregime eingehalten wird. Die Zählimpulse müssen dabei nicht völlig gleichmäßig ausgelöst, sondern können in ihrem zeitlichen Abstand an das Prozeßgeschehen angepaßt werden.



Zähl-impuls Nr.	Bitbelegung am Zählerausgang	La 6	La 5	La 4	La 3	La 2	La 1	Zeit-abschnitt
0	0 0 0 0							t_0
1	0 0 0 1						•	t_1
2	0 0 1 0					•		t_2
3	0 0 1 1				•			t_3
4	0 1 0 0			•				t_4
5	0 1 0 1		•					t_5
6	0 1 1 0	•						t_6
7	0 1 1 1	•						t_7
8	1 0 0 0	•	•					t_8
9	1 0 0 1	•	•	•				t_9
10	1 0 1 0	•	•	•	•			t_{10}
11	1 0 1 1	•	•	•	•	•		t_{11}
12	1 1 0 0	•	•	•	•	•	•	t_{12}
13	1 1 0 1			•	•			t_{13}
14	1 1 1 0							t_{14}
15	1 1 1 1			•	•			t_{15}

Bild 3.4. Anwendung des Bitmustergenerators zur Steuerung von wandernden Leuchtpunkten

3.2. Programmierbarer Funktionsgenerator

Ein programmierbarer Funktionsgenerator kann durch die Zusammensetzung eines Bitmustergenerators mit einem steuerbaren Spannungsstufengenerator realisiert werden.
Der steuerbare Spannungsstufengenerator formt aus Bitmustern Gleichspannungen mit einem vom Bitmuster abhängigen Spannungswert. Er ist ein Digital-Analog-Umsetzer (DAU).

Bild 3.5 zeigt das Blockbild der gesamten Anordnung. Darin erkennen wir den Bitmustergenerator und den DAU, der die Taktimpulse über ein Verzögerungsglied G5 etwas später erhält.

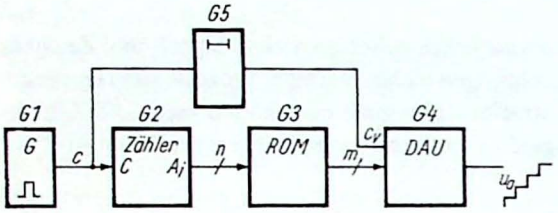


Bild 3.5
Programmierbarer Spannungsstufengenerator (Funktionsgenerator)

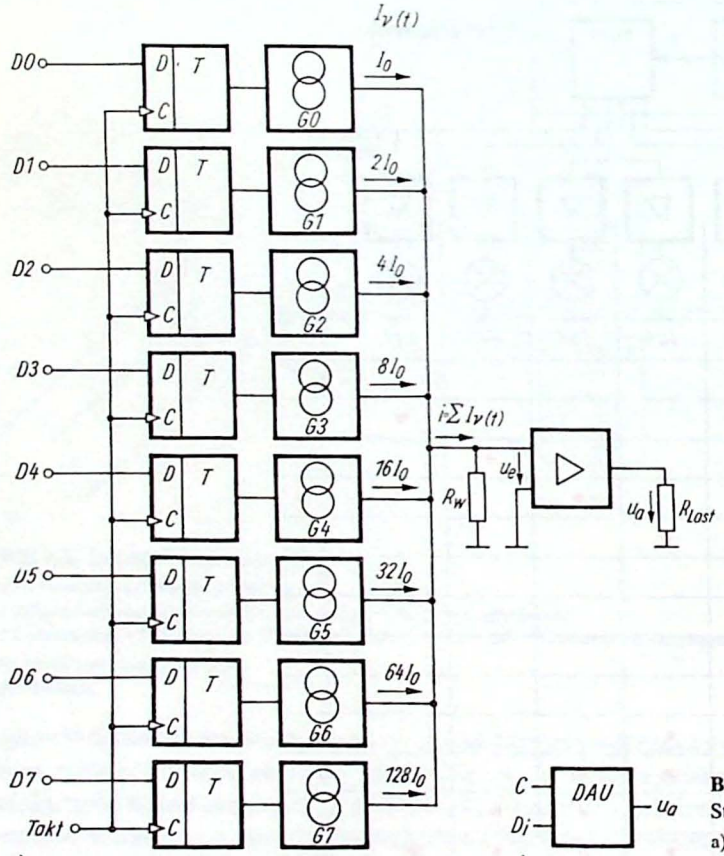


Bild 3.6
Steuerbarer Spannungsstufengenerator
a) Prinzipschaltbild; b) Schaltsymbol

Doch zunächst soll der steuerbare Spannungsstufengenerator erklärt werden **(Bild 3.6)**. Die Glieder G0 bis G7 stellen Konstantstromquellen dar, die durch ein 1-Signal (Zustand 1) an ihrem Eingang eingeschaltet und durch ein 0-Signal (Zustand 0) ausgeschaltet werden können. Im Widerstand R_w wird die Summe aller jeweils eingeschalteten Stromstärkewerte wirksam. Das Besondere

an dieser Anordnung ist, daß die von den Konstantstromquellen gelieferten Stromstärken nach dem Dualzahlenschema gestuft sind. Das erkennt man daran, daß sich von oben nach unten gliedweise die Stromstärke verdoppelt.

So wie wir mit 8 Bit jede mögliche ganze Zahl von 1 bis 255 darstellen können, ist mit dieser Anordnung von Stromquellen jedes ganzzahlige Stromstärkevielfache von I_0 bis 255 I_0 herstellbar.

Will man z. B. den Stromstärkewert 100 mA in R_w auftreten lassen und hat man $I_0 = 1$ mA gewählt, so müssen bestimmte Konstantstromquellen an ihrem Eingang das 1-Signal erhalten. *Vielleicht überprüfen Sie selbst, ehe Sie weiterlesen, welche Stromquellen das sein müssen.*

Die Lösung lautet: 1-Signal erhalten G6, G5, G2.

Im Widerstand R_w wird die auftretende Stromstärke in eine Spannung umgewandelt. Der nachfolgende Verstärker sorgt für eine Entkopplung zwischen R_{Last} und R_w, wobei die Ausgangsspannung U_a proportional zur Eingangsspannung U_e ist. Wählt man $R_w = 1\ \Omega$ und $v = 10$, so wird bei $I = 100$ mA eine Ausgangsspannung $U_a = 1$ V erzeugt.

Die taktflankengesteuerten D-Trigger, die vor die einzelnen Konstantstromquellen geschaltet sind, dienen zur Speicherung des zur Erzeugung einer bestimmten Stromstärke notwendigen Bitmusters.

Wir wollen uns nun die Aufgabe stellen, daß die im **Bild 3.7** unten dargestellte Spannungsstufenfunktion erzeugt werden soll. Dazu muß der ROM mit Hilfe der schwarz gezeichneten Taktim-

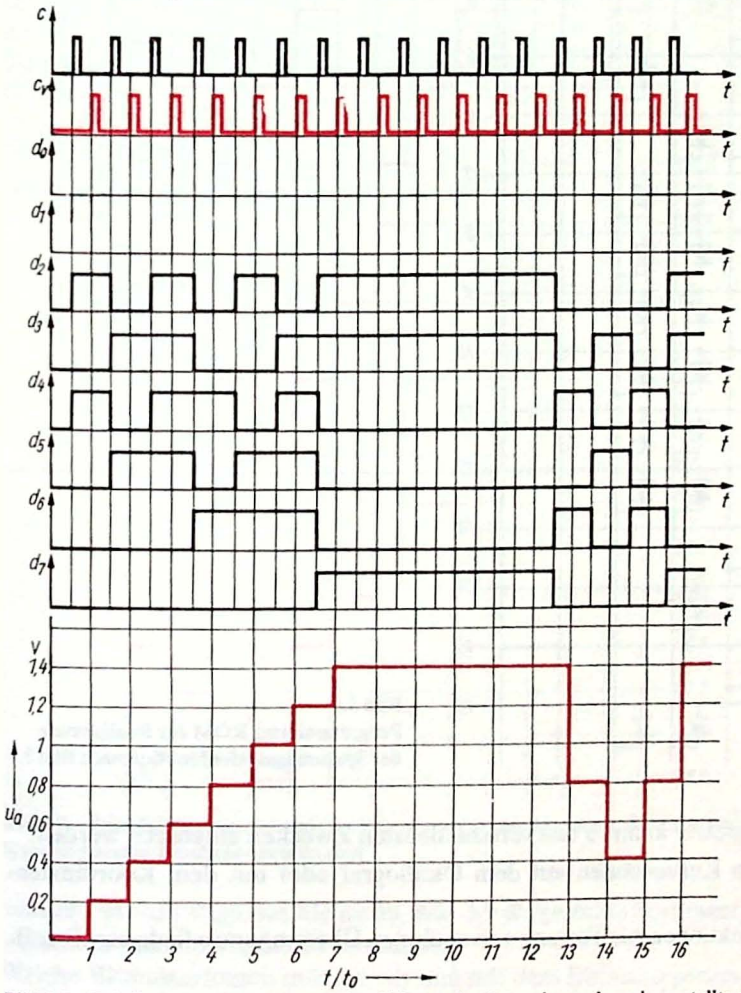


Bild 3.7. Zeitdiagramme zur Veranschaulichung des Entstehens einer dargestellten Spannungsstufenfunktion

pulse, durch deren Zählung die ROM-Adressen gebildet werden, das in den Teildiagrammen für d_0 bis d_7 gezeichnete Impulsmuster liefern. Im **Bild 3.8** ist der für einen einfachen ROM zur Lösung dieser Aufgabe notwendige Signalflußplan dargestellt.
Die Diodenverteilung im ROM stellt, wie wir wissen, das Abbild der Bitmuster dar. So kann man z. B. in Zeile 7 erkennen, daß Dioden für D7, D3 und D2 gesetzt sind und somit zu Beginn des Taktes 7 die Konstantstromquellen G7, G3 und G2 eingeschaltet, also insgesamt 140 mA geliefert werden. Dadurch entsteht am Ausgang des Spannungsstufengenerators eine Spannung von 1,4 V.

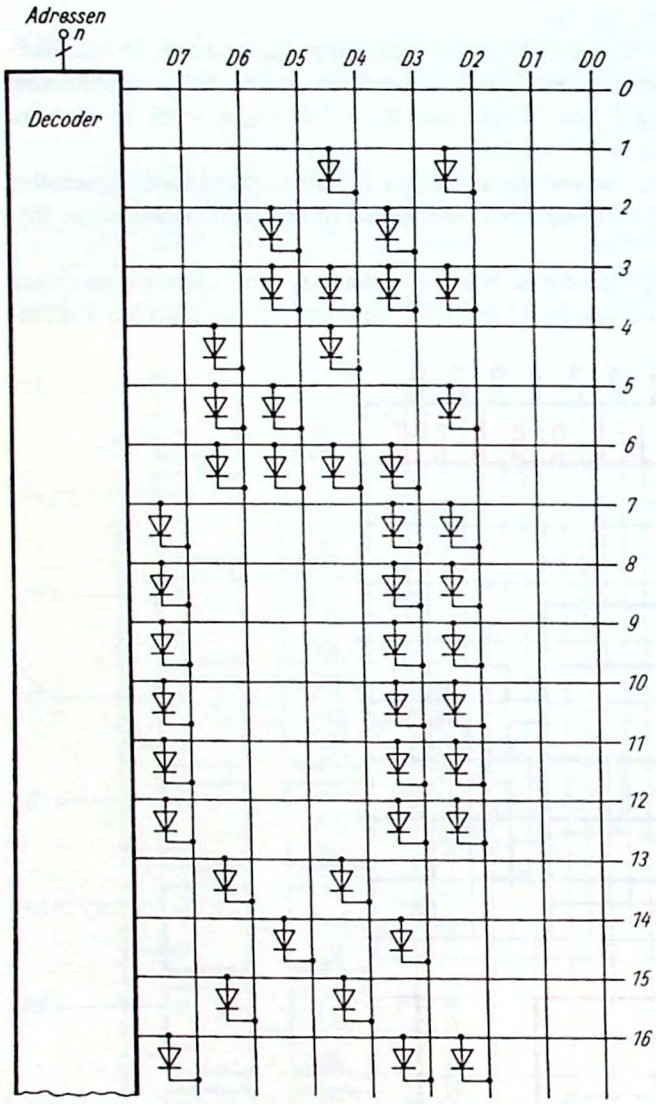


Bild 3.8
Programmierter ROM zur Realisierung der Spannungsstufenfunktion nach Bild 3.7

Ein solcher Digital-Analog-Umsetzer kann zu den verschiedensten Zwecken eingesetzt werden:

- Erzeugung von gewünschten Kurvenläufen auf dem Oszillograf oder mit dem Koordinatenschreiber
- Erzeugung beliebiger Zeitfunktionen zur Testung von analogen Übertragungsgliedern, wie z. B. analogen Regelkreisen
- Steuerung von analog stellbaren elektromechanischen Antrieben.

3.3. Mustergenerator

Der programmierbare Funktionsgenerator, den wir im vorigen Abschnitt kennengelernt haben, kann beliebige Stufenfunktionen erzeugen. Man muß nur dafür sorgen, daß die für eine gewünschte Funktion zu den einzelnen Spannungsstufen benötigten Bitmuster in der richtigen Reihenfolge im ROM oder RAM abgespeichert sind.

Wir wollen diese Problematik etwas erweitern und uns mit der Aufgabe beschäftigen, wie man beliebig viele Muster und vor allem solche mit Wiederholfolgen am zweckmäßigsten erzeugt.

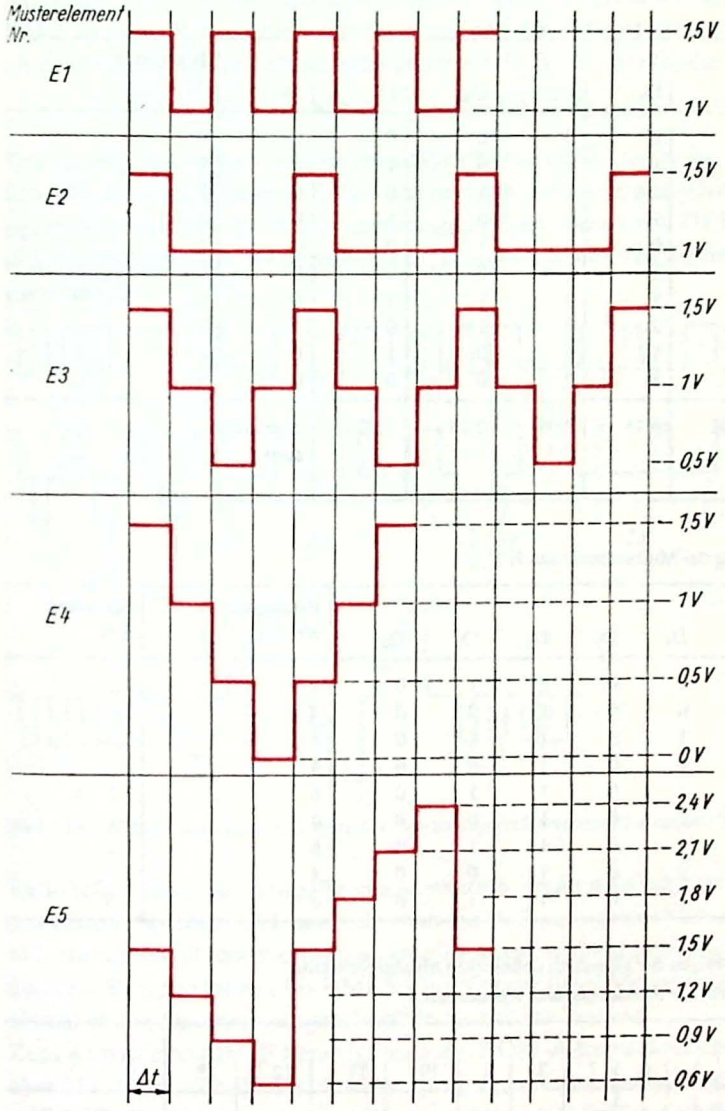


Bild 3.9. Willkürlich vorgegebenes Sortiment von Musterelementen, die aus einzelnen Spannungsstufen zusammengesetzt sind

Schauen wir uns dazu das kleine im **Bild 3.9** dargestellte Sortiment von Musterelementen an, das willkürlich zusammengestellt worden ist.

Welche Bitmusterfolgen müssen wir nun mit dem Bitmustergenerator bilden, um die Musterelemente zu erzeugen?

Bei Auswertung aller im Bild 3.9 dargestellten Muster ist zu ersehen, daß nur folgende Spannungsstufen benötigt werden:

0 V, 0,5 V, 0,6 V, 0,9 V, 1 V, 1,2 V, 1,5 V, 1,8 V, 2,1 V, 2,4 V.

Bleiben wir bei der im vorigen Abschnitt vorgegebenen Zuordnung, nach der 1 V der Stromstärke 100 mA entspricht und die Elementarstromstärke $I_0 = 1$ mA beträgt, so ergibt sich die in **Tafel 3.2** dargestellte Zuordnungstabelle von Bitmuster zu Spannungswert.

Tafel 3.2. Zuordnung von Bitmuster zu Spannungswert für die zur Erzeugung der Muster nach Bild 3.9 benötigten Spannungsstufen

U in V	Bitmuster								Bitmuster-Nr.
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	0	0	0	0	0	0	0	0	0
0,5	0	0	1	1	0	0	1	0	1
0,6	0	0	1	1	1	1	0	0	2
0,9	0	1	0	1	1	0	1	0	3
1	0	1	1	0	0	1	0	0	4
1,2	0	1	1	1	1	0	0	0	5
1,5	1	0	0	1	0	1	1	0	6
1,8	1	0	1	1	0	1	0	0	7
2,1	1	1	0	1	0	0	1	0	8
2,4	1	1	1	1	0	0	0	0	9
Spannungswert der Bitstelle in V	1,28	0,64	0,32	0,16	0,08	0,04	0,02	Markierspur	

Tafel 3.3. Bitmusterfolge zur Erzeugung des Musterelementes E 1

Taktimpuls-Nr.	Bitmuster								Bitmuster-Nr.	Spannung in V
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
1	1	0	0	1	0	1	1	0	6	1,5
2	0	1	1	0	0	1	0	0	4	1
3	1	0	0	1	0	1	1	0	6	1,5
4	0	1	1	0	0	1	0	0	4	1
5	1	0	0	1	0	1	1	0	6	1,5
6	0	1	1	0	0	1	0	0	4	1
7	1	0	0	1	0	1	1	0	6	1,5
8	0	1	1	0	0	1	0	0	4	1
9	1	0	0	1	0	1	1	0	6	1,5

Tafel 3.4. Übersicht über die Bitmusterfolgen der einzelnen definierten Musterelemente
(Die einzelnen Bitmuster werden nur noch durch ihre Bitmusternummer repräsentiert.)

	Taktstufe													
	1	2	3	4	5	6	7	8	9	10	11	12	13	
Bitmuster Nr.	6	4	6	4	6	4	6	4	6					BMF1 ≙ E1
Bitmuster Nr.	6	4	4	4	6	4	4	4	6	4	4	4	6	BFM2 ≙ E2
Bitmuster Nr.	6	4	1	4	6	4	1	4	6	4	1	4	6	BFM3 ≙ E3
Bitmuster Nr.	6	4	1	0	1	4	6							BFM4 ≙ E4
Bitmuster Nr.	6	5	3	2	6	7	8	9	6					BFM5 ≙ E5

Um nun beispielsweise das Musterelement E1 zu erzeugen, ist die in **Tafel 3.3** dargestellte Bitmusterfolge erforderlich. Das Bitmuster der Tafel 3.3 wird dazu im ROM des Bitmustergenerators abgespeichert. Ähnliche Tabellen und zugehörige ROM-Platzbelegungen erhält man für die anderen Musterelemente.
Zur leichteren Überschaubarkeit wollen wir bei der weiteren Betrachtung nicht mehr mit dem jeweiligen Muster der Bitstellen D0 bis D7, sondern nur noch mit der zugehörigen Bitmuster-Nr. arbeiten. Mit diesen Nummern erhält man dann die in **Tafel 3.4** für jedes Musterelement notierte Zahlenfolge, die sich wesentlich leichter lesen läßt.
Anschließend wollen wir aus den Musterelementen irgendwelche Muster zusammenstellen, wie sie **Bild 3.10** zeigt. Wir nehmen zunächst an, daß sich diese Muster, so wie sie dargestellt sind, beliebig oft wiederholen sollen. Dann erhalten wir z. B. für Muster Ⓑ die Musterelementfolge:

E3 E2 E4 E2 E3/E3 E2 E4 E2 E3/E3 E2 E4 E2 E3/.../

Würden wir die Folge so im ROM abspeichern, wie sie hier steht, dann wäre das sehr unwirtschaftlich. Für die eben gelesene Folge von drei gleichen Gruppen (Gruppe ≙ E3 E2 E4 E2 E3) benötigt man bereits 3 × 59 Takte; das entspricht der Anzahl von 177 Wörtern im ROM. Da das Muster sich aber beliebig oft wiederholen soll, müßten für jede Wiederholung weitere 59 Wörter abgespeichert werden.

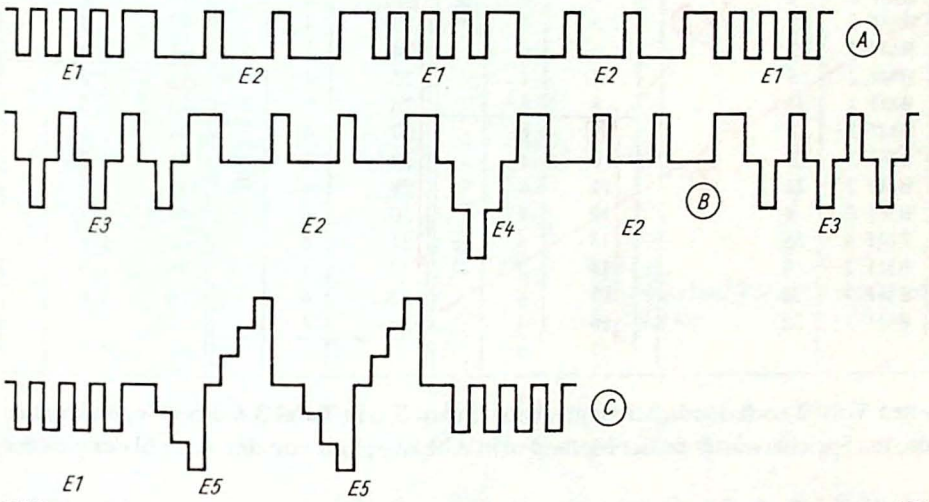


Bild 3.10. Willkürlich zusammengestellte Spannungsstufenmuster, die aus den Musterelementen nach Bild 3.9 bestehen

Rationeller kann die Aufgabe gelöst werden, wenn man mit zwei Speichern arbeitet **Tafel 3.5** : In den ersten, Speicher M1, schreibt man die Reihenfolge der Musterelemente; im zweiten, Speicher M2, stehen die Bitmusterfolgen der einzelnen definierten Musterelemente (abgekürzt: BMF), in unserem Beispiel BMF1 bis BMF5. Der Vorteil dieses Verfahrens besteht darin, daß jede BMF nur einmal in den Speicher eingeschrieben zu werden braucht.
Zum Aufruf einer BMF benötigt man die ROM-Adresse ihrer ersten Spannungsstufe, die im Speicher M1 steht. Von dieser Adresse aus folgen die nächsten Spannungsstufenadressen in geordneter, aufwärts zählender Reihenfolge, so daß zum Abruf der nächsten Stufen nur noch die Adresse jeweils um eins erhöht werden muß. Ist dann die gesamte BMF realisiert, muß ein Zeichen gegeben werden, daß der Vorgang beendet ist. Danach wird vom Speicher M1 die Information geholt, welches Musterelement als nächstes folgen soll.
Es ist leicht einzusehen, daß nach dieser Verfahrensweise der Speicherplatzbedarf zur Erzeugung des Musters B wesentlich geringer geworden ist als bei der zuvor zitierten Methode, bei der alle benötigten BMF immer wieder erneut im ROM eingeschrieben werden mußten. Während, wie oben

berechnet, nach dieser ursprünglichen Methode für die ersten drei Mustergruppen ein Speicherplatzbedarf von 177 Wörtern entstand, sind es nach der neuen Methode nur noch 48 Wörter. Nach der ursprünglichen Methode erhöhte sich mit jeder Wiederholung der Zeichenfolge E3, E2, E4, E2, E3 die benötigte Speicherplatzanzahl um 59 Wörter. Nach der neuen Methode erhöht sie sich nur um 5 Wörter, die zur Ausgabe der Anfangsadressen der jeweiligen BMF im Speicher M1 benötigt werden.

Tafel 3.5. Schematische Darstellung der Speicherplatzbelegung zur Erzeugung des Musters B nach dem Zweispeicherverfahren

Speicher M1			Speicher M2									
	Daten (D)											
Adresse ADR	Name	Anfangsadresse in M2	ADR	D		ADR	D		ADR	D		
0	–	–	0	6		18	4		35	6		
1	BMF 3	22	1	4		19	4		36	4		
2	BMF 2	9	2	6	BMF 1	20	4		37	1		BMF 4
3	BMF 4	35	3	4		21	6		38	0		
4	BMF 2	9	4	6		22	6		39	1		
5	BMF 3	22	5	4		23	4		40	4		
6	BMF 3	22	6	6		24	1		41	6		
7	BMF 2	9	7	4		25	4		42	6		
8	BMF 4	35	8	6		26	6	BMF 3	43	5		
9	BMF 2	9	9	6		27	4		44	3		
10	BMF 3	22	10	4		28	1		45	2		
11	BMF 3	22	11	4		29	4		46	6		BMF 5
12	BMF 2	9	12	4	BMF 2	30	6		47	7		
13	BMF 4	35	13	6		31	4		48	8		
14	BMF 2	9	14	4		32	1		49	9		
15	BMF 3	22	15	4		33	4		50	6		
16	BMF 3	22	16	4		34	6					
⋮	⋮	⋮	17	6								

Um den erzielten Vorteil noch deutlicher zu machen, finden Sie in **Tafel 3.6** eine Gegenüberstellung der benötigten Speicherwörter beider Methoden in Abhängigkeit von der Anzahl der gleichen Gruppen.

Wir wollen nun die vorteilhafte neue Methode begrifflich besser fassen.

Die Aufteilung in zwei Speicher entspricht einer Abarbeitung in zwei Ebenen. Im Speicher M1 steht das sog. *Hauptprogramm*, und im Speicher M2 stehen die *Unterprogramme*.

Tafel 3.6. Gegenüberstellung des Bedarfs an Speicherwörtern für die ursprüngliche, lineare Abarbeitungsweise und für das Zweispeicherverfahren in Abhängigkeit von der Anzahl der Wiederholungen des Musters (Anzahl gleicher Gruppen)

Anzahl der gleichen Gruppen	Benötigte Speicherwörter	
	ursprüngliche Methode	neue Methode
1	59	38
2	118	43
3	177	48
4	236	53
5	295	58
10	590	83
100	5900	533

Die Abarbeitung beginnt im Hauptprogramm durch Aufruf der ersten Adresse, die im Beispiel durch die Zahl 1 gegeben ist. Auf diesem Speicherplatz wird die Zahl 22 gefunden und als Adresse für den Speicher M2 benutzt. Dadurch kann M2 das auf dem Speicherplatz mit der Adresse 22 befindliche Datenwort ausgeben, was in unserem Fall dem Bitmuster 6 entspricht, das nach Tafel 3.2 1001 0110 lautet.

Danach wird das Unterprogramm Wort für Wort abgearbeitet, bis im Speicher M2 die Adresse 34 erreicht wird und damit die BMF3 realisiert ist. Nach Abarbeitung dieses auf Adresse 34 stehenden Wortes wird in das Hauptprogramm zurückgesprungen und der Inhalt des Speicherplatzes mit der Adresse 2 gelesen. Dieser Inhalt lautet 9 und dient als Adresse für den Speicher M2. Es erfolgt nun die Abarbeitung der BMF2 im Speicher M2, bis dort die Adresse 21 erreicht wird. Dann geht es wieder zurück in das Hauptprogramm, um dort die nächste Unterprogramm-Anfangsadresse zu holen. **Bild 3.11** zeigt Ihnen den gesamten Ablauf in Form eines *Programmablaufplans*, in dem aber ausnahmsweise nur Adressen von Speicherplätzen notiert sind. Um die Speicherplatzadressen unterscheiden zu können, bekommen die Adressen des Hauptprogramms den Vorsatz H und die der Unterprogramme den Vorsatz U.

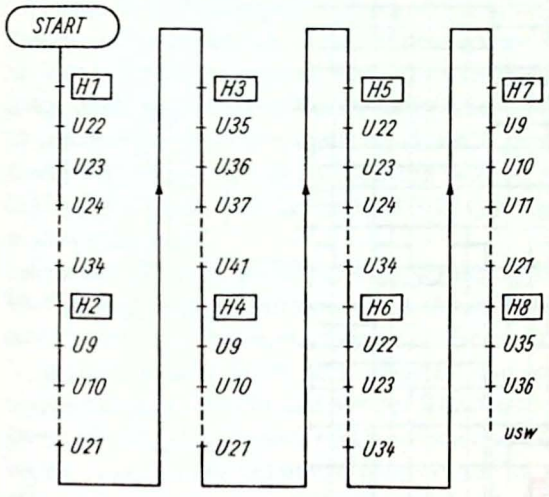


Bild 3.11
Programmablaufplan zur Erzeugung des Musters B nach Bild 3.10

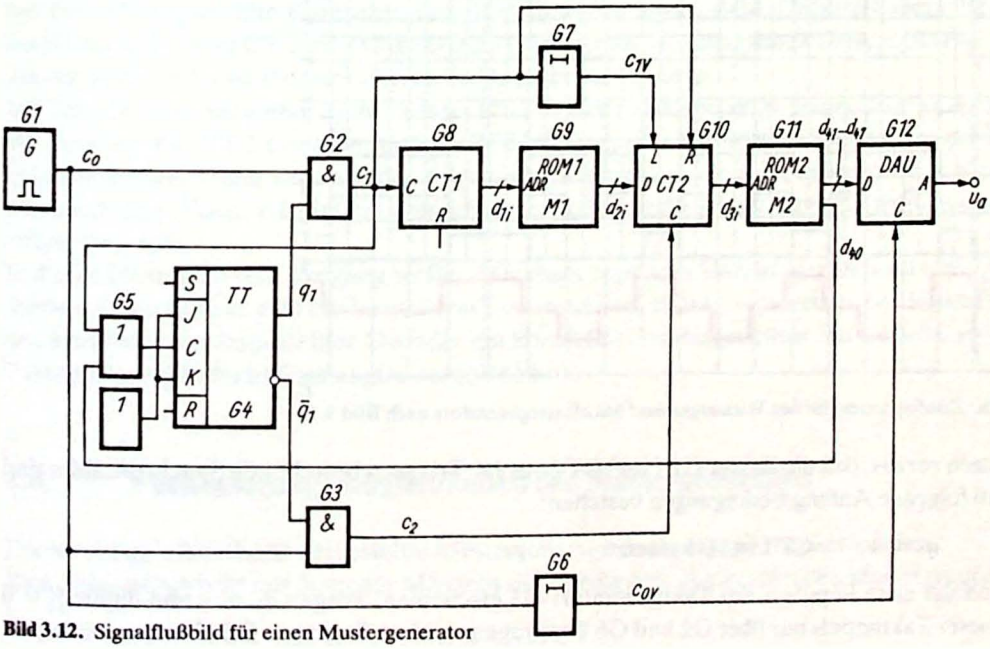


Bild 3.12. Signalflußbild für einen Mustergenerator

Nach diesen Betrachtungen zur Ablauforganisation wollen wir das Signalflußbild **(Bild 3.12)** und das zugehörige Zeitdiagramm **(Bild 3.13)** einer möglichen Realisierungsvariante kennenlernen. Das im Bild 3.12 dargestellte System besteht aus den beiden Speichern ROM 1 (Hauptprogrammspeicher) und ROM 2 (Unterprogrammspeicher), den Zählern CT 1, CT 2, dem Spannungsstufengenerator DAU und einigen weiteren Logikgliedern.

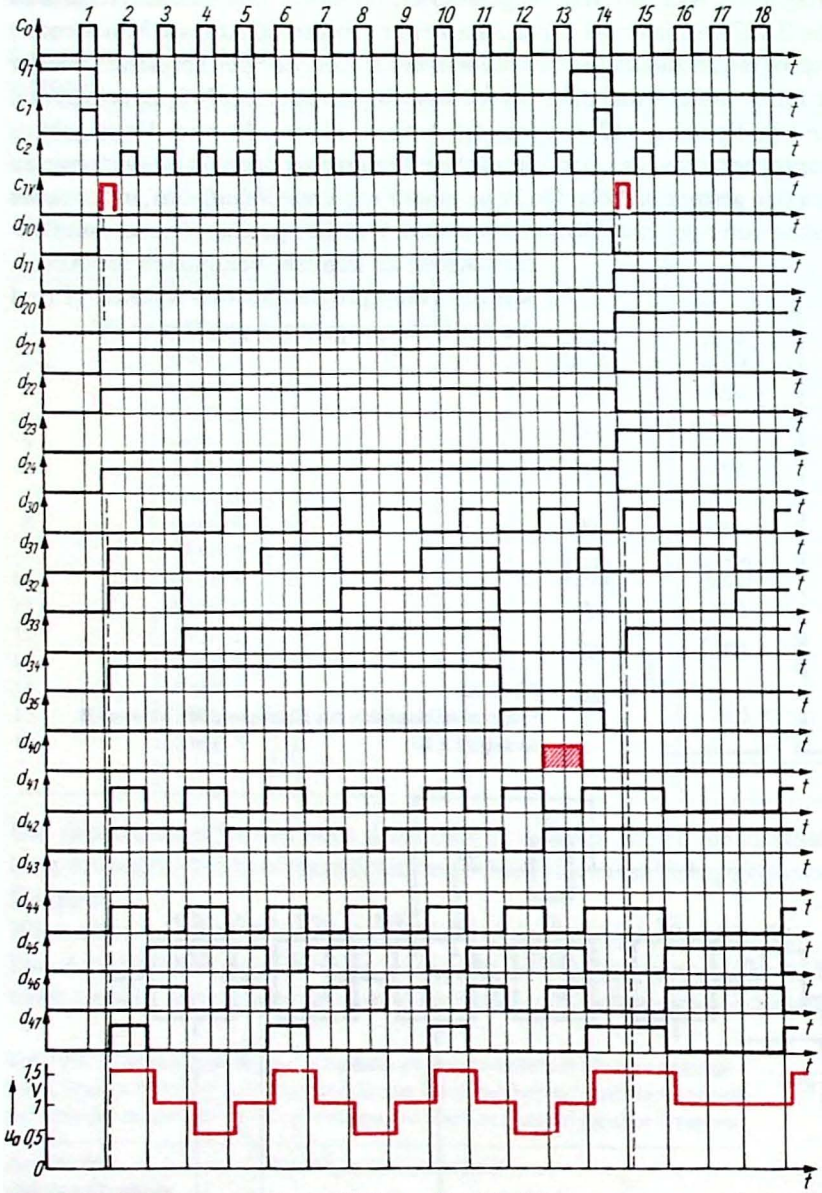


Bild 3.13. Zeitdiagramme für den Wirkungsablauf des Mustergenerators nach Bild 3.12

Wir setzen voraus, daß die Zähler CT 1 und CT 2 aus JK-Triggern bestehende Synchronzähler sind und daß folgende Anfangsbedingungen bestehen:

$$q_1 = 1, \qquad \text{CT 1 ist rückgesetzt.}$$

Jetzt soll der erste Impuls c_0 des Taktgenerators G1 erscheinen. Wegen $q_1 = 1$ und damit $\bar{q}_1 = 0$ kann dieser Taktimpuls nur über G2 und G6 übertragen werden. Der über G2 übertragene Impuls

stellt den Zähler CT 2 zurück und erhöht den Stand des Zählers CT 1 um eins. Damit wird ab 1/0-Flanke des Taktimpulses das erste Wort im ROM 1 (Anfangsadresse der gewählten BMF) an die Dateneingänge des Zählers CT 2 gelegt. Mit dem über G7 zeitlich verzögerten Taktimpuls wird die Information an den Dateneingängen in den Zähler übernommen, so daß dieser Zählerstand im ROM 2 die Adresse bilden kann. Diese schaltet das damit aufgerufene Wort des ROM 2 auf den Eingang des Spannungsstufengenerators DAU.
Übernommen wird die Information in den DAU erst, wenn die über G6 verzögerte 0/1-Flanke des Taktimpulses wirksam wird. Diese Verzögerung muß so groß gewählt werden, daß in allen Fällen der DAU erst dann übernimmt, wenn die Informationen des ROM 2 vorliegen.
Jetzt haben wir noch den Weg des Taktimpulses c_1 über G5 und G4 zu verfolgen. Mit der 0/1-Flanke von c_1 wird der JK-Trigger G4 bereits auf Umschalten in den anderen Zustand vorbereitet, und mit der 1/0-Flanke von c_1 übernimmt der Slave den neuen Zustand $q_1 = 0, \bar{q}_1 = 1$. Damit wird das Tor G2 geschlossen und Tor G3 geöffnet.
Alle folgenden Taktimpulse können jetzt nur noch über G3 den Zählerstand von CT 2 erhöhen. Das ist aber beabsichtigt; denn mit dem ersten Wort von ROM 1 haben wir die Anfangsadresse der gewählten BMF (im Beispiel BMF3) vorgegeben, und nun werden aufwärts zählend die anderen zu dieser BMF gehörenden Wörter abgefordert. Mit dem zweiten Taktimpuls wird im CT 2 die duale 23, mit dem dritten Taktimpuls die duale 24 gebildet.
Damit werden die in Tafel 3.5, rechte Seite ab Adresse 22 stehenden Daten in Binärform auf den DAU gegeben, so daß das im Bild 3.13 unten gezeichnete Muster entsteht, das den Anfang vom Muster B darstellt.
Interessant wird das Zeitdiagramm wieder ab Taktimpuls 12. Hier ist das vorletzte Bitmuster der BMF3 gerade aufgerufen worden. Auf der Datenleitung D40 des ROM 2 ist hier ein Markierimpuls (rot markiert) programmiert worden, der zum Umschalten des JK-Triggers G4 dient. Nach Tafel 3.2 ist Bit D0 (hier D40) als Markierbit vorgesehen. Mit der Vorderflanke des Markierimpulses wird der Master von G4 und mit der Rückflanke der Slave umgeschaltet.
Damit ist das Tor G2 wieder freigegeben und Tor G3 gesperrt. Man kann erkennen, daß der Taktimpuls 13 noch über G3 übertragen worden ist, so daß nun die BMF3 vollständig abgearbeitet ist. Das letzte Bitmuster von Takt 13 wird durch die Verzögerung durch G6 so lange gehalten, bis das mit Takt 14 ausgewählte Bitmuster vom ROM 2 zur Verfügung steht. Dadurch kommt das infolge des Rücksetzens von CT 2 im Taktübergang (Taktimpuls 14) entstehende Datenmuster d_{41} bis d_{47}, das der ROM-Adresse 0 entspricht, im DAU nicht zur Wirkung.
Mit Takt 14 wird dann über c_1 der Zähler CT 1 um eins weitergetaktet, so daß nun die duale Zwei am Ausgang von CT 1 erscheint und im ROM 1 die duale Neun (die Anfangsadresse der BMF2) aufgerufen wird. Dann wird mit den folgenden Taktimpulsen die Adresse im ROM 2, ausgehend von der dualen Neun, ständig um eins erhöht, so daß jetzt die BMF2 Wort für Wort an den DAU ausgegeben wird.
In dieser Form geht der Vorgang weiter. An einen regulären Halt ist hierbei noch nicht gedacht worden. Dazu könnte man ein besonderes Codewort vom ROM 1 verabreden, bei dessen Erscheinen ein hier nicht dargestellter Decoder ein ebenfalls nicht dargestelltes Tor schließt, so daß die Taktimpulse nicht mehr übertragen werden können.

3.4. Verbesserungsmöglichkeiten des Mustergenerators

Der im vorigen Abschnitt dargestellte Mustergenerator kann noch verbessert werden.
Zum Beispiel braucht der Speicher M1 nicht die ständig sich wiederholenden Mustergruppen

E3, E2, E4, E2, E3/E3, E2, E4, E2, E3/...

bzw., genauer gesagt, die ständig sich wiederholende Anfangsadressenfolge der eingesetzten BMF_i

22, 9, 35, 9, 22/22, 9, 35, 9, 22/...

zu enthalten. Es genügt, wenn man diese Folge nur einmal speichert und dann angibt, wie oft diese Folge wiederholt werden soll. Dazu benötigt man dann nur einen auf die Anzahl der Wiederholungen voreinstellbaren Rückwärtszähler, dessen Zählerstand nach jedmaligem Abarbeiten der fünf Adressen (22, 9, 35, 9, 22) um eins vermindert wird und der nach Erreichen des Standes Null mit Hilfe eines Decoders und einer Torschaltung den Halt herbeiführt. Der Programmablaufplan bekommt dann die Form nach **Bild 3.14**.

Andere Verbesserungen sind möglich, wenn man folgendes bedenkt: Die eben entworfene *Wiederholungsschleife* erfordert die Einbeziehung eines zusätzlichen Zählers im Signalflußplan nach Bild 3.12. Nun könnte auch der Fall eintreten, daß sich eine Folge von einzelnen Musterkombinationen, wie

E1 E2 E3/E1 E2 E3/E1 E2 E3/E4 E1/ E4 E1/.../ E1 E2 E3/E1 E2 E3/
E1 E2 E3/E4 E1/E4 E1/...,

insgesamt z. B. zehnmal wiederholt. Dann könnte man mit weiteren Programmschleifen arbeiten **(Bild 3.15)**.

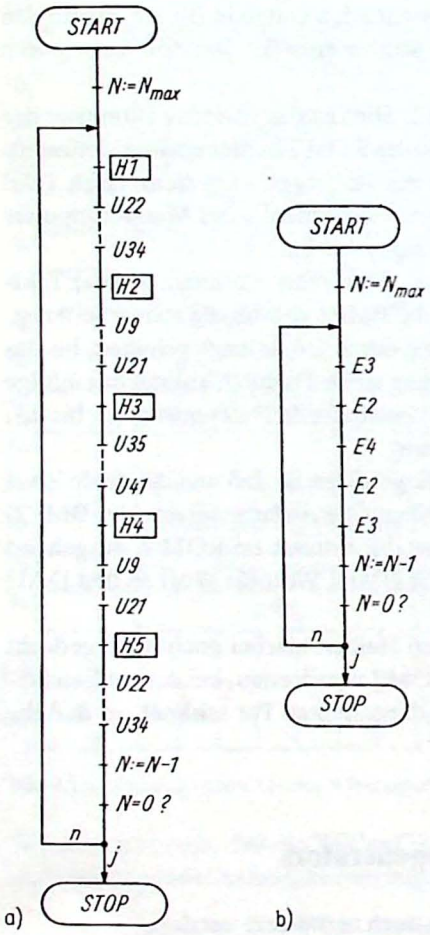


Bild 3.14. Zyklischer Programmablaufplan zur Erzeugung des Musters B nach Bild 3.10
a) ausführliche Darstellung; b) verkürzte Darstellung

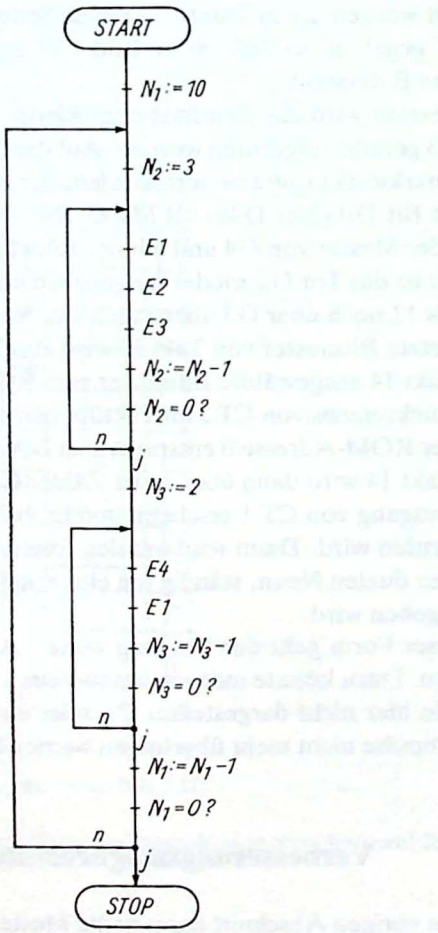


Bild 3.15. Programmablaufplan zur Erzeugung eines Musters, das mehrere Programmschleifen erfordert

Die Realisierung dieser Schleifenbildung ist sicher sehr effektiv. Aber es müssen jetzt insgesamt drei Zusatzzähler hinzugefügt werden. Außerdem ist zu bedenken, daß die bisher betrachteten Muster nur willkürlich herausgegriffene, einfache Beispiele sind und es noch wesentlich kompliziertere Anordnungen mit einer viel größeren Anzahl von Schleifen geben kann. Will man an solche vielseitige Mustergestaltung rasch anpassungsfähig sein, muß unbedingt ein anderes Prinzip für die Automatengestaltung angewendet werden. Anstelle der jeweils neu zu verdrahtenden Ausführung muß ein Universalautomat treten, der in einfacher Form auf neue Probleme angepaßt werden kann. Die Schaltung muß immer die gleiche bleiben, aber der Informationsfluß und die Reihenfolge der Informationsverarbeitungsschritte müssen leicht veränderlich sein.
Das ist ein ähnliches Problem wie in einem Produktionsbetrieb. Hat man eine fest eingestellte automatische Fertigung, so erfordert eine Umstellung auf neue Produkte um so mehr Kosten und Mühe, je mehr die Maschinen auf eine bestimmte Funktion spezialisiert sind. Hat man dagegen programmierbare Universalmaschinen, die nach Programmen oder Zeichnungsvorlage jedesmal ein völlig anderes Teil herstellen können, so genügt zur Umstellung die Eingabe dieser Informationen.
Der Universalautomat muß also vielseitige Fertigkeiten bezüglich der Verarbeitung von Informationen aufweisen und sich schnell auf die nächsten Forderungen umstellen lassen. Diesem System muß der Mensch relativ einfach die Arbeitsvorschriften überreichen können.
Wir werden sehen, daß die Grundlage eines solchen Automaten in Form des Digitalrechners bereits entwickelt worden ist. Mit den Technologien der Mikroelektronik sind dabei Rechner entstanden, die auf Grund ihrer im Mikrometerbereich liegenden Bauelementeabmessungen als *Mikrorechner* bezeichnet werden. Doch bevor wir uns diesen universellen Steuer- und Rechenbausteinen zuwenden, wollen wir anhand eines leichter überschaubaren Beispiels einen wichtigen Teil dieser Rechner kennenlernen: das Matrixsteuerwerk.

3.5. Matrixsteuerwerk

Ein Matrixsteuerwerk entspricht in seiner wesentlichen Funktion einer programmierbaren Logikanordnung (PLA) mit Rückkopplung.

Zum besseren Verständnis seiner Wirkungsweise wollen wir die Lösung der folgenden Aufgabenstellung als mögliches Anwendungsbeispiel heranziehen:
Ein Meßgerät soll von einem Meßobjekt in bestimmten Zeitabständen Meßwerte ermitteln. Um nicht alle kleinen Schwankungen der Meßwerte zu erfassen, soll von einer bestimmten Anzahl n aufeinanderfolgender Meßwerte m_i jeweils der arithmetische Mittelwert gebildet und an ein Erfassungsgerät ausgegeben werden.
Der arithmetische Mittelwert $\overline{m_v}$ berechet sich zu

$$\overline{m_v} = \frac{m_1 + m_2 + m_3 + \ldots + m_n}{n} = \frac{1}{n} \sum_{i=1}^{n} m_i.$$

Um mit der Division durch n keine besonderen Schwierigkeiten zu bekommen, wollen wir für n nur folgende Werte zulassen:

$$n = 2, 4, 8, 16, 32 = 2^1, 2^2, 2^3, 2^4, 2^5.$$

Alle zugelassenen Werte sind durch Potenzieren der Zahl zwei mit ganzen Zahlen berechenbar. Wie wir von den Seiten 15, 109 wissen, kann die Division einer Dualzahl durch eine Potenzzahl zur

Basis zwei durch Rechtsverschiebung gelöst werden. Dazu muß die zu dividierende Zahl um so viele Stellen nach rechts verschoben werden, wie es der Potenzexponent angibt.

Bild 3.16 zeigt das Blockbild des Schaltungsentwurfs. Hierbei bedienen wir uns einer stark zusammengefaßten Blockdarstellung, weil eine weitere Vereinzelung eine große Darstellungsfläche erfordert und relativ unübersichtlich würde. Rot gezeichnet sind die vom Steuerwerk ausgehenden Steuerleitungen bzw. die zum Steuerwerk hinführenden Informationsleitungen. Der Durchlauf der Meßinformation ist durch schwarze Pfeile gekennzeichnet.

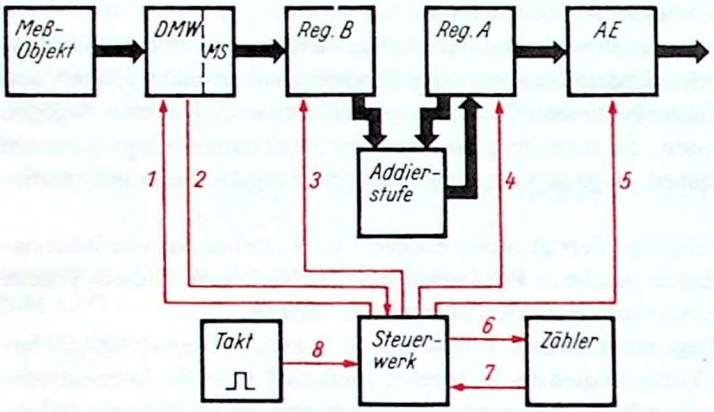


Bild 3.16. Blockbild zur Realisierung der arithmetischen Mittelwertbildung von Meßwerten
DMW digitalisierender Meßwertaufnehmer
MS Meßwertspeicher
AE Ausgabeeinheit

Das Meßobjekt ist irgendein konkretes Objekt eines technischen Prozesses, an dem eine bestimmte physikalische Größe (Spannung, Stromstärke, Anpreßdruck, Kraft, Geschwindigkeit oder anderes) beobachtet werden soll.

Der digitalisierende Meßwertaufnehmer (DMW) muß in der Lage sein, den Zahlenwert der physikalischen Größe als Dualzahl bereitzustellen und diese Dualzahl in einem Meßwertspeicher (MS) aufzubewahren. Über die Steuerleitung 1 wird dem DMW die Aufforderung vom Steuerwerk übermittelt, die nächste Messung vorzunehmen. Über die Leitung 2 teilt der DMW dem Steuerwerk mit, daß das Meßergebnis im Meßwertspeicher abgelegt ist.

Vom Meßwertspeicher gelangt der Meßwert in das Reg. B. Dazu sendet das Steuerwerk über Leitung 3 ein Signal, das die Übernahme der Meßinformation durch Reg. B auslöst.

Das Reg. A nimmt, ausgehend vom Anfangszustand 0, das Ergebnis der Addition auf und läßt den neuen Inhalt von Reg. B jeweils dazu addieren. Auf Grund dieser Eigenschaft wird es als *Akkumulator* (Sammler) bezeichnet.

Nachdem alle für die jeweilige Mittelwertbildung erforderlichen Meßwerte im Akkumulator aufsummiert sind, muß die Rechtsverschiebung um die benötigte Stellenzahl erfolgen, damit die Division durch die entsprechende Zweierpotenz realisiert wird. Danach kann das Ergebnis an die Ausgabeeinheit (AE) übergeben werden.

Das diesen Prozeß kommandierende Steuerwerk erhält seine Zeitangaben durch den Taktgenerator. Es stellt den Zähler (wir arbeiten hier zur Programmschleifenbildung nochmals mit einem Zähler!) auf die Anzahl der abzuarbeitenden Schleifen ein, sendet dem Zähler die Zählimpulse und wird von ihm benachrichtigt, wenn die Schleifenabarbeitung beendet ist.

Wir wollen voraussetzen, daß der Speicher MS und alle Register aus solchen Speicherelementen (z. B. D-Triggern) aufgebaut sein sollen, die keine Löschung des vorhergehenden Inhalts bei Paralleldateneingabe erforderlich machen.

Im **Bild 3.17** ist der Programmablaufplan und im **Bild 3.18** das zugehörige Zeitdiagramm dargestellt (s. auch Tafel 3.7). Aus Übersichtsgründen wurde nur eine Mittelwertbildung über je zwei Meßwerte veranschaulicht. *Versuchen Sie nun, anhand dieses Programmablaufplans die Zusammenhänge zu erkennen, und prüfen Sie, ob Sie die bisherige Ablaufbeschreibung verstanden haben.* Dabei bedeutet z die vom Zählerstand dargestellte Dualzahl. Die winkligen Klammern um Reg. A, Reg. B, AE bedeuten, daß nicht das durch die Buchstaben gekennzeichete Register an sich, sondern dessen eingespeicherte Dualzahl gemeint ist. Man liest ⟨Reg. A⟩ als Inhalt vom Register Reg. A.

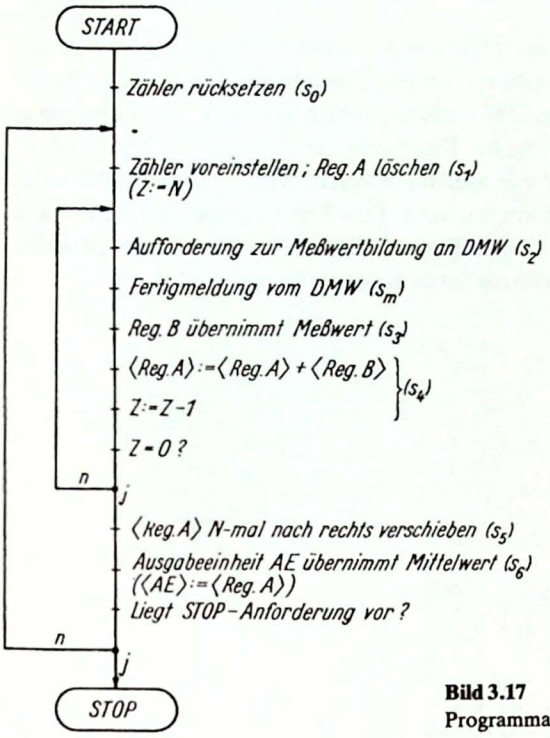


Bild 3.17
Programmablaufplan zum Blockbild nach Bild 3.16

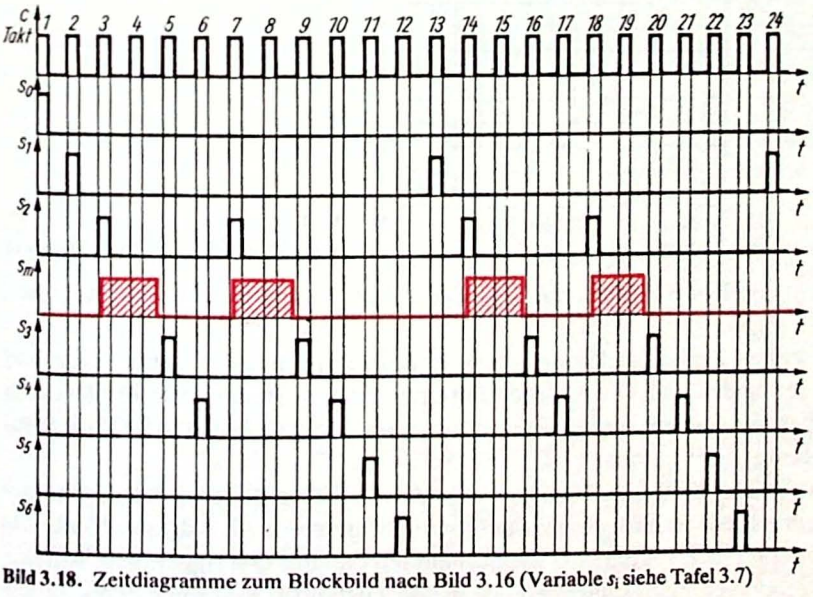


Bild 3.18. Zeitdiagramme zum Blockbild nach Bild 3.16 (Variable s_i siehe Tafel 3.7)

Im Zeitdiagramm (Bild 3.18) ist durch die Taktzeiteinteilung eine größere Durchschaubarkeit entstanden als im Bild 3.13. Das liegt daran, daß nahezu alle Vorgänge taktsynchron verlaufen, d. h. mit den Taktimpulsen zeitlich zusammen ausgelöst werden. Das Steuerwerk verteilt sozusagen die Taktimpulszeit auf die einzelnen Glieder, die jeweils betätigt werden sollen.
Es gibt nur noch einen Vorgang, der nicht taktsynchron (asynchron) verläuft. Das ist die Beendigung des Meßvorgangs. Dieser Zeitpunkt kann natürlich nur vom Meßgerät selbst bestimmt werden. Dabei ist in unserem Fall angenommen worden, daß der digitalisierende Meßwertaufnehmer (DMW) in der Zeit von der Aufforderung zur Messung bis zur Fertigmeldung selbständig mit eigener Taktsteuerung gearbeitet hat.
Wenn Sie sich jetzt den Ablauf des Zeitdiagramms mit Hilfe des Programmablaufplans verständlich machen, werden Sie einsehen, wie sehr sich beide ergänzen. Im Programmablaufplan erfolgt die Ablaufplanung ohne konkrete Zeitpunktfestlegung. Das Zeitdiagramm päzisiert den Programmablaufplan und stellt daher eine weitere Verfeinerung des Entwurfs dar.
Der Übersichtsgewinn, der durch die taktsynchrone Steuerung erreicht wird, läßt sich noch weiter vergrößern, wenn man das Steuerwerk in Matrixform entwirft. Das Prinzip eines solchen Matrixsteuerwerks wurde bereits 1951 von *M. V. Wilkes* vorgeschlagen. Dieses Matrixschema entspricht einer PLA, die taktgesteuert in sich selbst jeweils die nächsten Informationen findet.

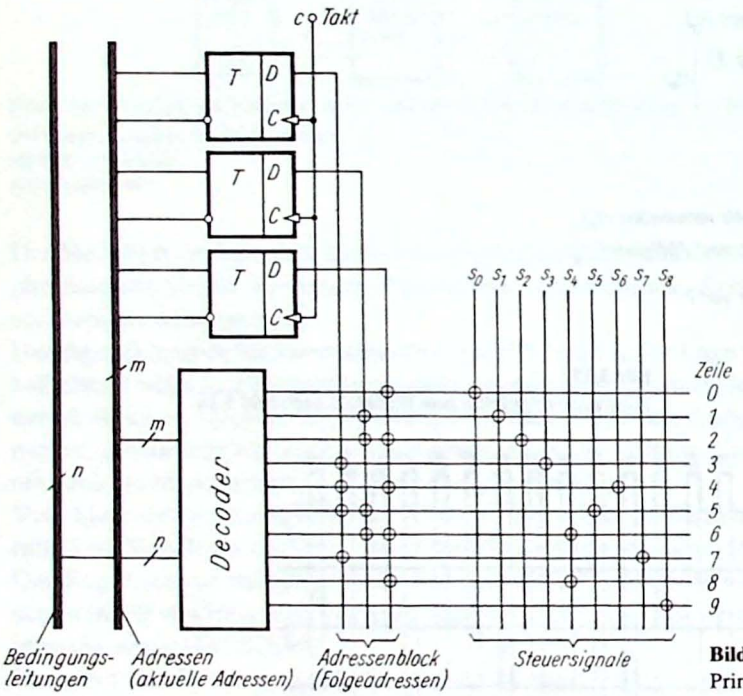


Bild 3.19
Prinzipschaltbild eines Matrixsteuerwerks

Bild 3.19 zeigt das Prinzip und **Bild 3.20** einen Signalflußplan zur Lösung unserer Aufgabenstellung.
Die im Bild 3.19 dargestellten Kreise markieren solche Kreuzungspunkte, in denen Zeile und Spalte durch eine Diode verbunden sind. Die Abarbeitung eines solchen Steuerwerks erfolgt durch taktsynchrones Umschalten von einer aktivierten Zeile zu einer anderen, wobei in jeder Zeile Steuersignale und Folgeadressen erzeugt werden.
Nehmen wir an, daß sich zunächst Zeile 0 im Zustand 1 befindet. In dieser Zeile wird der Steuerimpuls s_0 erzeugt und die Adresse der als nächste anzusteuernden Matrixzeile im Adressenblock vorbereitet. Erscheint dann der nächste Taktimpuls, so übernehmen die drei D-Trigger diese Adresse und schalten über den Decoder die auszuwählende Zeile in den Zustand 1. Das soll hier die Zeile 1

sein. Dadurch wird der Steuerimpuls s_1 erzeugt und die Adresse der nächsten, zu aktivierenden Zeile bereitgestellt.

Durch die Bedingungsleitungen können je nach dem Zustand, den bestimmte, ausgewählte Glieder der Schaltung melden, bei gleichen Adressen auch andere Zeilen ausgewählt werden. So werden z. B. in den Zeilen 0 und 8 gleiche Adressen bereitgestellt, die eigentlich auf die gleiche Zeile führen müßten. Durch zusätzliche Berücksichtigung des Zustands einer Bedingungsleitung bei der Decodierung kann jedoch eine Aufteilung in zwei verschiedene Zeilen erreicht werden, so daß mit dieser Maßnahme eine *Programmverzweigung* realisiert werden kann.

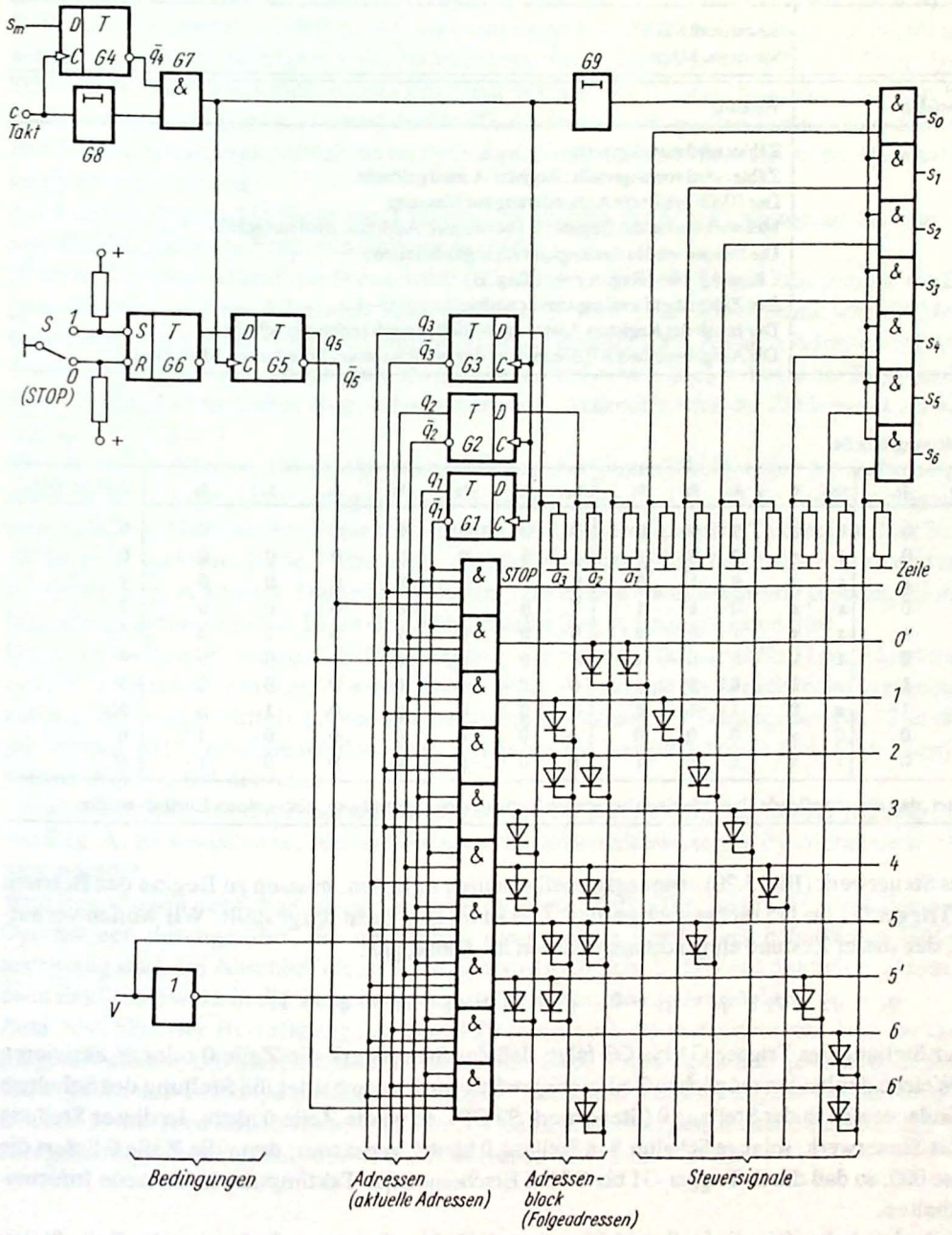


Bild 3.20. Programmiertes Matrixsteuerwerk zur Bildung des arithmetischen Mittelwerts von Meßwerten

Doch gehen wir jetzt zur Betrachtung des Bildes 3.20 über. Um diesen Signalflußplan voll verstehen zu können, benötigen wir noch das Blockbild 3.16, den Programmablaufplan (Bild 3.17), das Zeitdiagramm (Bild 3.18) und die **Tafel 3.7**.

Tafel 3.7. Variablendefinition und Schaltbelegungstabelle für das Matrixsteuerwerk nach Bild 3.20

Statische Variable	Zustand
$v = 0$	Zählerstand $z = 0$
$v = 1$	Zählerstand $z > 0$
$q_5 = 0$	Steuerwerk STOP
$q_5 = 1$	Steuerwerk läuft
Impulsvariable	**Wirkung**
s_0	Zähler wird zurückgesetzt
s_1	Zähler wird voreingestellt; Register A wird gelöscht
s_2	Der DMW erhält die Aufforderung zur Messung
s_3	Meßwert wird in das Register B übertragen; Addition wird ausgeführt
s_4	Die Summe wird in das Register A eingeschrieben; $\langle \text{Reg. A} \rangle := \langle \text{Reg. A} \rangle + \langle \text{Reg. B} \rangle$ Der Zählerstand wird um eins erniedrigt; $z := z - 1$
s_5	Der Inhalt des Registers A wird um N Stellen nach rechts verschoben
s_6	Die Ausgabeeinheit AE übernimmt den gebildeten arithmetischen Mittelwert

Schaltbelegungstabelle

q_3	q_2	q_1	q_5	v	a_3	a_2	a_1	s_0	s_1	s_2	s_3	s_4	s_5	s_6	Aktive Zeile
0	0	0	0	x	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	x	0	0	1	1	0	0	0	0	0	0	0′
0	0	1	x	x	0	1	0	0	1	0	0	0	0	0	1
0	1	0	x	x	0	1	1	0	0	1	0	0	0	0	2
0	1	1	x	x	1	0	0	0	0	0	1	0	0	0	3
1	0	0	x	x	1	0	1	0	0	0	0	1	0	0	4
1	0	1	x	1	0	1	1	0	0	1	0	0	0	0	5
1	0	1	x	0	1	1	0	0	0	0	0	0	1	0	5′
1	1	0	0	x	0	0	0	0	0	0	0	0	0	1	6
1	1	0	1	x	0	0	1	0	0	0	0	0	0	1	6′

x bedeutet, daß die betreffende Variable einen beliebigen Zustand einnehmen kann, also keinen Einfluß ausübt

Um das Steuerwerk (Bild 3.20) ordnungsgemäß anlaufen zu lassen, müssen zu Beginn des Betriebs die D-Trigger G1 bis G5 rückgesetzt werden. Das ist im Bild nicht dargestellt. Wir wollen voraussetzen, daß dieser Zustand eingenommen worden ist. Damit gilt:

$$q_1 = q_2 = q_3 = q_4 = q_5 = 0, \quad \bar{q}_1 = \bar{q}_2 = \bar{q}_3 = \bar{q}_4 = \bar{q}_5 = 1.$$

Aus der Stellung der Trigger G1 bis G3 folgt, daß das Steuerwerk die Zeile 0 oder 0′ aktivieren muß. Welche der beiden möglichen Zeilen ausgewählt wird, entscheidet die Stellung des Schalters S. Befindet er sich in der Stellung 0 (Steuerwerk STOP), so ist die Zeile 0 aktiv. In dieser Stellung muß das Steuerwerk, solange Schalter S in Stellung 0 bleibt, verharren; denn die Zeile 0 liefert die Adresse 000, so daß die D-Trigger G1 bis G3 bei Erscheinen der Taktimpulse keine neue Information erhalten.
Wird jedoch Schalter S in die Stellung 1 (Steuerwerk läuft) gebracht, so bekommt die Zeile 0′ den

Zustand 1 und Zeile 0 den Zustand 0. In der Zeile 0′ werden das Steuersignal s_0 (Rücksetzen des Zählers) und die Adreßinformation 001 gebildet.
Beim Eintreffen des nächsten Taktimpulses wird Zeile 1 aktiviert, das Steuersignal s_1 (Voreinstellen des Zählers und Löschen des Registers A) gegeben und die Adresse 010 vorbereitet.
Mit dem Taktimpuls Nr. 3 wird Zeile 2 aktiv, das Steuersignal s_2 (Aufforderung zur Messung) gegeben und die Adresse 011 gebildet. Der digitalisierende Meßwertaufnehmer, der die Aufforderung zur Meßwertbildung erhalten hat, muß nun die Variable s_m in den Zustand 1 versetzen und diesen Zustand so lange halten, bis das Meßergebnis im Meßwertspeicher MS vorliegt. Durch $s_m = 1$ geschieht folgendes: Mit der positiven Flanke des nächsten Taktimpulses (im Beispiel Nr. 4) wird der D-Trigger G4 gesetzt, so daß $\bar{q}_4 = 0$ wird und damit Tor G7 schließt. Da der Taktimpuls über G8 entsprechend lange verzögert wird, findet er keinen Durchlaß mehr durch Tor G7. Damit bleibt das Steuerwerk ohne Taktimpulsversorgung und verharrt im Wartezustand, der erst dann beendet wird, wenn $s_m = 0$ gegeben ist und die 0/1-Flanke des darauffolgenden Taktimpulses erscheint.
Durch diese Schaltungsmaßnahme wird die asynchron erfolgende Beendigung der Meßwertbildung taktsynchron wirksam.
Auch der D-Trigger G5 dient zur Taktsynchronisierung der Schaltvorgänge am Schalter S, während G6 die bekannte Entprellschaltung darstellt.
Doch zurück zum Ablauf des Steuerwerks. Zuletzt war die Zeile 2 aktiv. Mit dem auf die Beendigung des Meßvorgangs folgenden Taktimpuls (Nr. 5) wird Zeile 3 aktiv. Damit wird die Meßinformation nach Reg. B übernommen und die Addition durchgeführt. Als neue Adresse wird 100 gebildet. Mit dem folgenden Taktimpuls (Nr. 6) wird Zeile 4 aktiv. Mit $s_4 = 1$ wird das Ergebnis der Addition in den Akkumulator Reg. A eingeschrieben. Außerdem wird der Zählerstand um eins vermindert: $z := z - 1$.
Mit der neuen Adresse 101 erfolgt beim nächsten Taktimpuls (Nr. 7) wegen der Bedingung $z \neq 0$ und damit $v = 1$ die Aktivierung der Zeile 5. Es wird mit $s_2 = 1$ erneut die Aufforderung zur Messung gegeben. Nach abgeschlossener Messung wird mit dem nächsten Taktimpuls (hier Nr. 9) die Adresse 011 wirksam. Zeile 3 wird aktiv, die Meßinformation zum Reg. B übertragen und mit dem Inhalt von Reg. A addiert. Damit ist die Summe der beiden ersten Meßwerte gebildet, die mit dem folgenden Taktimpuls (Nr. 10) in den Akkumulator Reg. A eingeschrieben wird.
Die nun angebotene Adresse 101 führt wegen $z = v = 0$ mit Taktimpuls Nr. 11 zur Aktivierung der Zeile 5′. Der Inhalt von Reg. A wird dadurch mit $s_5 = 1$ um eine Stelle nach rechts geschoben. Damit liegt der fertige Mittelwert vor und kann nach Erscheinen des Taktimpulses Nr. 12 an die Ausgabeeinheit AE übergeben werden. Ist $q_5 = 1$ geblieben, so wird dabei die Zeile 6′ aktiviert, die als nächste Adresse 001 anbietet.
Damit wiederholt sich das Programm ab der erneuten Voreinstellung des Zählers und dem Löschen von Reg. A. Es werden anschließend die nächsten beiden Meßwerte und ihr arithmetischer Mittelwert gebildet.
Wird irgendwann der Schalter S in die Stellung 0 (Steuerwerk STOP) gebracht, so werden noch alle Operationen durchgeführt, die zur Bildung des gerade in Bearbeitung befindlichen Mittelwerts notwendig sind. Im Abschluß dieser Aktionen wird nach Zeile 5′ die Zeile 6 aktiviert. Danach läuft dann das Steuerwerk in die Zeile 0, die den stabilen Wartezustand ermöglicht.
Zum Abschluß der Betrachtung von Bild 3.20 sollen noch die taktgesteuerten Tore für die Ausgangsvariablen s_i erläutert werden. Diese dienen dazu, Steuerimpulse zu gewinnen, deren Zeitdauer gleich der Taktimpulsdauer ist. Durch die zusätzliche Verzögerung des Taktimpulses durch G9, ehe er die Tore öffnen kann, wird erreicht, daß alle im Übergangszustand auftretenden Schaltunregelmäßigkeiten (spikes) des Matrixsteuerwerks unwirksam bleiben.

3.6. Auf dem Wege zum universellen Steuersystem

Die im vorigen Abschnitt behandelte Mittelwertsbildung von Meßwertfolgen führte uns auf einen Automaten, der mit einem Mikroprozessor schon eine gewisse Ähnlichkeit aufzuweisen hat. Dieser Meßautomat ist allerdings recht einseitig nur für eine einzige Aufgabe ausgelegt. Er stellt eine Spezialentwicklung dar.
Will man solche Spezialentwicklungen wirtschaftlich auf einem Halbleiterchip fertigen, müssen Stückzahlen von mehr als 10000 garantiert sein. Eine solch große Anzahl von Anwendungsfällen läßt sich für manche Aufgaben nicht nachweisen, so daß diese dann nicht in dieser Minimalform gelöst werden könnten, wenn nicht Universalbausteine entwickelt worden wären:

Mikroprozessor und Mikrorechner.

Beide entstammen einer technischen Idee, die die Menschen schon seit Jahrhunderten beschäftigt: dem Rechenautomat.

Von der ersten Recheneinrichtung, dem *Abakus*, der seit mehr als 3000 Jahren bekannt ist, über mechanische Addiereinrichtungen gab es bis zur heutigen modernen Rechentechnik eine Vielzahl von Entwicklungsstufen.
Während man beim Abakus auf Drähten sitzende Kugeln verschiebt, aber bereits mit Übertrag addieren und subtrahieren kann, wurden Anfang des 17. Jahrhunderts die ersten mechanischen Addiermaschinen mit Zahnrädern entwickelt. 1671 wurde von *Leibniz* ein mechanischer Stufenrechner vorgestellt, mit dem schon multipliziert werden konnte. 1833 entwarf *Charles Babbage* einen mechanischen Rechenautomaten, der bereits die Programmierung und auch Speichereinheiten vorsah, der aber wegen der ungenügend ausgereiften Technologie nicht vollendet wurde.
Die ersten 1937–1945 gebauten elektrischen, programmgesteuerten Rechenautomaten hatten ein fest verdrahtetes Programm, das bei Programmänderungen jedesmal neu verdrahtet werden mußte. Einen großen Aufschwung in der elektronischen Rechentechnik gab es deshalb 1944, als *v. Neumann* ein Prinzip der Programmspeicherung ohne Verdrahtungsmaßnahmen vorschlug. 1951 machte *Wilkes* das Prinzip des Matrixsteuerwerks bekannt, von dem wir im vorigen Abschnitt ein Beispiel kennengelernt haben. Voll wirksam wurde dieses übersichtliche Steuerwerk erst mit der Verbilligung aller Elemente durch die Mikroelektronik.
Wie sich unter dem Einfluß der Verkleinerung der räumlichen Abmessungen aller Bauelemente und der Senkung des Energiebedarfs die Rechenautomaten entwickelt haben, wurde bereits auf S. 50, Tafel 2.14, dargestellt.
Wie kommt es nun dazu, daß dieser Rechenautomat als Grundlage eines universellen Steuerungssystems verwendet werden konnte?
Zuerst wurden die Digitalrechner zum Lösen komplizierter mathematischer Probleme (z. B. Berechnung optischer Systeme) oder zur Massenberechnung von Routineaufgaben (z. B. Lohn- und Gehaltsabrechnungen, Versicherungsberechnungen, Produktions- und Umsatzberechnungen) verwendet. Doch dann stellte man die Frage, warum ein solches System nur für Zahlenrechnungen verwendet werden soll?
Sehr vereinfacht gesehen, übt doch der Digitalrechner folgende Funktion aus:

- Ein eingegebenes Bitmuster wird nach einer Bearbeitungsvorschrift (Programm) in ein Ausgangsbitmuster umgewandelt.

Genau das aber wird auch von allen anderen Logiksystemen realisiert.
Im Abschnitt 1. haben wir gesehen, daß bei bewußt definierten Zeichen alles nur eine Frage der Verabredung ist. Ein Bitmuster kann unterschiedliche Sachverhalte ausdrücken. Es kann Zustandsangaben aller Art beinhalten, wie z. B.

- Mengenangaben (reine Zahlenwerte)
- Größenangaben (Zahlenwert × Maßeinheit)
- graduell gestufte Angaben
  (z. B. Härteskala, musikalische Tempobezeichnungen)
- Prozeßzustände
- Aussagen beliebiger Art.

Sie werden nun vielleicht sagen, daß alles soweit einzusehen ist, was das Bitmuster betrifft. Aber sind nicht die Bearbeitungsvorschriften grundsätzlich verschieden?
Im Prinzip schon. Doch jede Bearbeitungsvorschrift kann in einzelne Elemente (Teiloperationen) aufgegliedert werden. Und auf dieser Ebene zeigen sich Gemeinsamkeiten.
Betrachten wir zunächst nur die Rechenoperationen. Diese lassen sich alle auf die Elementaroperationen *Addition* und *Subtraktion* zurückführen. Die *Multiplikation* ist eine fortgesetzte Addition, die *Division* eine fortgesetzte Subtraktion. Integral- und Differentialrechnung werden über Differenzen und Differenzenquotienten auf die Elementaroperationen zurückgeführt. Jede Funktion läßt sich durch Potenzreihen und diese wiederum durch die Elementaroperationen ausdrücken.
Also benötigt ein Digitalrechner nur die Elementaroperationen Addition und Subtraktion, um sämtliche mathematischen Operationen ausführen zu können.
Nehmen wir als nächstes die logischen Operationen, mit deren Hilfe man Probleme der Schaltalgebra (Boolesche Algebra), aber auch der Aussagelogik bearbeiten kann. Hier können ebenfalls alle Operationen auf wenige Elementaroperationen (UND, ODER, Negation) zurückgeführt werden.
Verbleiben noch die Aussagen beliebiger Art, die man im einfachsten Fall auf die Bitstelle genau nimmt. Dann ist ihre Identifizierung mit einem Suchprozeß unter bekannten Begriffen verbunden. Man muß dann in der *Liste* der bekannten Begriffe suchen, welcher davon mit dem gelieferten Bitmuster identisch ist. Das führt auf die Elementaroperation *Bitmustervergleich.*
Eine mögliche Realisierung besteht darin, die Bitmuster Bit für Bit mit der Antivalenzfunktion (EXOR) zu verknüpfen. Erscheint im Ergebnis dieser Operation eine Bitbelegung, die der Zahl null entspricht, waren die Bitbelegungen in allen Stellen gleich. Sind die Bitmuster identisch, müssen es auch die zugehörigen Begriffe sein.
Wir sehen also, daß es Elementaroperationen gibt, die man nur kombinieren muß, um Informationen beliebiger Art verarbeiten zu können. Es muß allerdings dazu gesagt werden, daß die Kombination der Elementaroperationen zu funktionsfähigen Bearbeitungsvorschriften oftmals große Anstrengungen erfordert. Infolge der abnehmenden Preise für mikroelektronische Recheneinheiten werden heute die Kosten eines Systems maßgeblich von den Programmierungskosten bestimmt.
Die Eigenschaft des Digitalrechners, schon die zur Lösung beliebiger Informationsverarbeitungsaufgaben benötigten Elementaroperationen ausführen zu können, war der eine Grund, weshalb man dieses System zum universellen Steuersystem heranzog. Der andere Grund besteht im Vorhandensein einer hochentwickelten Verfahrensweise zur Schaffung beliebiger Arbeitprogramme (s. S. 184 ff.).
Die Arbeitsprogramme enthalten in geordneter Reihenfolge alle zur Realisierung eines bestimmten Steuerablaufs notwendigen Anweisungen in binärcodierter Form. Man spricht vom *Makroprogramm.* Der digitale Rechenautomat liest nun der Reihe nach jede einzelne Anweisung des Makroprogramms ein. Durch jeden Befehl des Makroprogramms wird das Programmsteuerwerk des Rechners so beeinflußt, daß die zugehörige, den Befehl realisierende Folge von Mikrooperationen erzeugt wird. Die Folge der Mikrooperationen wird analog als *Mikroprogramm* bezeichnet.
Diese Verfahrensweise entspricht im Prinzip dem im Abschnitt 3.3. auf den Seiten 137 und 138 vorgestellten 2-Speicher-Verfahren. Dort hatten wir im Speicher M2 die Musterelemente notiert. Dieser Speicher entspricht vergleichsweise dem *Mikroprogrammspeicher.* Der Speicher M1 enthielt die Anweisungen, in welcher Reihenfolge die in M2 gespeicherten Musterelemente erscheinen sol-

len. Er entspricht dem *Makroprogrammspeicher* (auch Hauptprogrammspeicher). Daß diese Verfahrensweise sehr effektiv ist, haben wir im Abschnitt 3.3. bewiesen.
Wenn man nun noch den Speicher M2 als Matrixsteuerwerk nach dem Prinzip von *M. V. Wilkes* ausführt, können die immer wiederkehrenden Elementaroperationen festverdrahtet im Speicher M2 abgelegt werden. Das hat den Vorteil, daß diese Elementaroperationen fest definiert sind und der Programmierer konstante und optimale Vorschriften für ihre Verwendung erhält.
Die Realisierung des Makro-/Mikroprogramm-Prinzips unter Einsatz des Matrixsteuerwerks von *M. V. Wilkes* war eine wesentliche Voraussetzung für ein entsprechend universell einsetzbares Informationsverarbeitungssystem.
Es war aber noch ein weiteres Problem auf dem Wege zum universellen Steuersystem zu lösen. Es ging dabei um die Frage, wie der Rechner zu seinen Informationen gelangt und in welcher Form er seine Arbeitsergebnisse ausgibt.
Man muß bedenken, daß der normale Digitalrechner – Sie können dazu Ihren Taschenrechner betrachten – seine Informationen durch Tasteneingabe vom Menschen erhält und seine Ergebnisse über eine Leuchtanzeige ausgibt. Der Mensch muß dann wieder die Ergebnisse ablesen und weiterverarbeiten. Ein technischer Prozeß, in dem ein Mensch die Rechenwerte abliest und dann vielleicht an einem Handrad die nötigen Steuerungen ausführt, dürfte in vielen Fällen recht unrentabel und zu langsam sein. Deshalb wird versucht, den Menschen aus diesem Geschehen herauszulösen und den vollautomatischen Prozeß zu verwirklichen. Das erfordert eine Vielzahl von Erfassungsgeräten und auch eine besondere Form der Rechenwertausgabe.

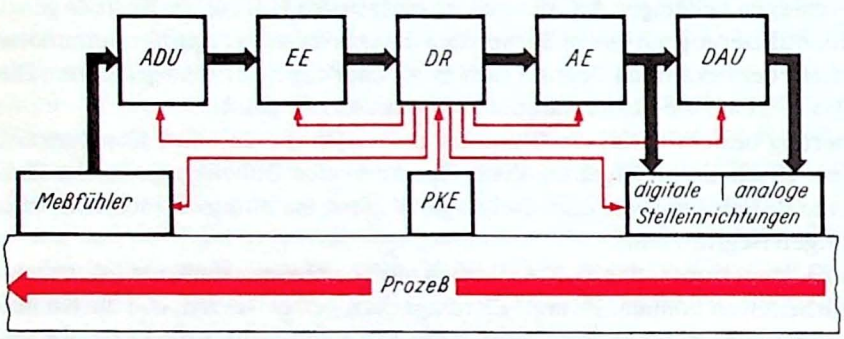


Bild 3.21. Blockbild eines automatischen Steuerungssystems bei Einsatz eines Digitalrechners
ADU Analog-Digital-Umsetzer; EE Eingabeeinheit; DR Digitalrechner; AE Ausgabeeinheit; DAU Digital-Analog-Umsetzer; PKE Prozeßkontrolleinheit

Wir wollen aus den notwendigen Erfordernissen heraus überlegen, welche Fuktionen von einem automatischen Steuersystem ausgeübt werden müssen. Aus Gründen der Übersichtlichkeit sollten wir wieder die Blockbildmethode mit der Darstellung grober Funktionsblöcke anwenden. **Bild 3.21** zeigt eine erste minimale Variante. **Bild 3.22** stellt das zugehörige Zeitdiagramm dar.
Im Bild 3.21 wird demonstriert, daß zunächst Meßinformationen (Meßwerte) durch einen Meßfühler (Sensor) gewonnen werden müssen. Dabei muß der Wert der Meßgröße, bevor er zum Rechner gelangt, in ein elektrisches Binärsignal (digitaler Spannungswert) umgewandelt werden. Wir wollen hier den Fall annehmen, daß die Meßwerte von einem analogen Größenverlauf stammen, also erst noch digitalisiert werden müssen. Im Bild 3.22 ist als analoge Meßgröße ein Druckverlauf dargestellt, der durch den Meßfühler in einen entsprechenden analogen elektrischen Spannungsverlauf umgewandelt wird. Der elektrische Spannungsverlauf wird anschließend in konstanten Zeitabständen abgetastet. Nachdem die Abtastwerte durch den Analog-Digital-Umsetzer (ADU) in elektrische Binärwerte umgesetzt sind, können sie über eine Eingabeeinheit (EE) dem Digitalrechner (DR) übergeben werden.

Der Digitalrechner kann nun mit Hilfe seines Programms den Meßwertverlauf analysieren und den optimalen Stellsignalverlauf zur Prozeßsteuerung ermitteln. Er gibt als Endprodukt digitale Stellwerte über die Ausgabeeinheit (AE) aus. Werden analoge Stelleinrichtungen verwendet, müssen aus den digitalen Stellwerten mit Hilfe eines Digital-Analog-Umsetzers (DAU) (s. S. 132 bis 134) noch diskrete Intensitätswerte (quasianaloge Werte) gebildet werden.

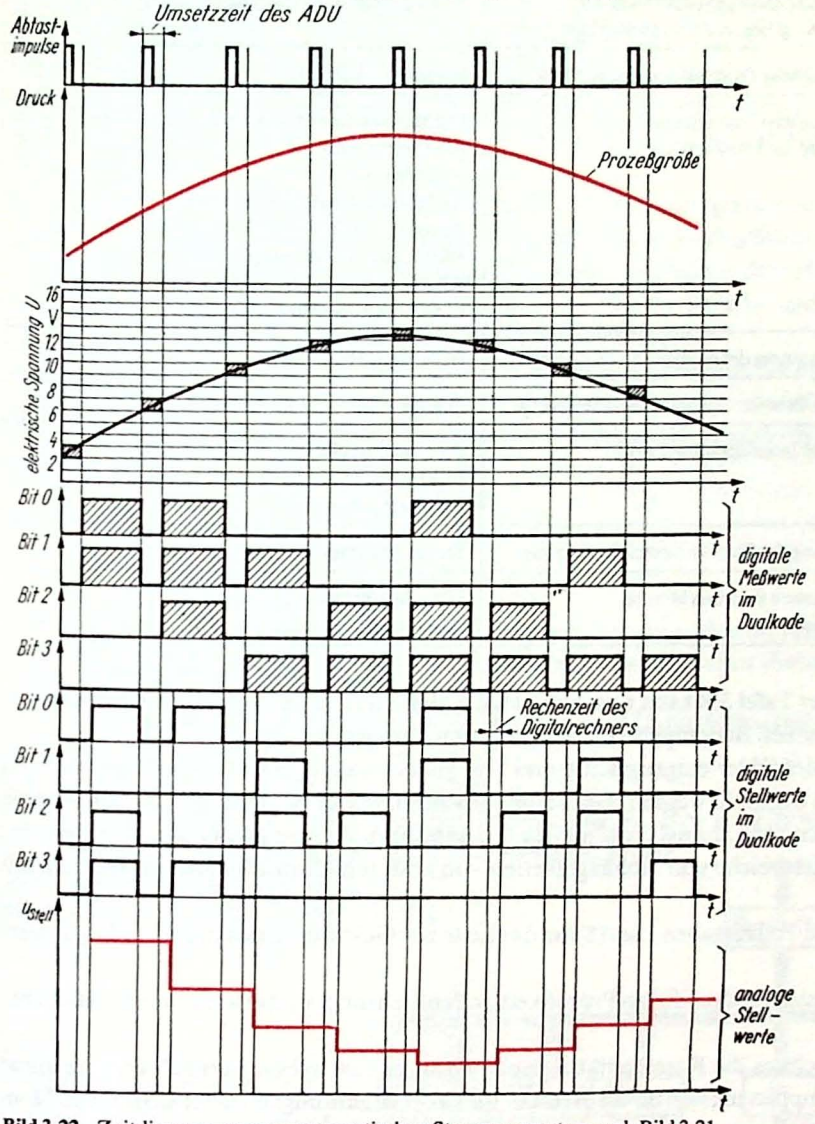


Bild 3.22. Zeitdiagramme zum automatischen Steuerungssystem nach Bild 3.21

Sollte der Gesamtprozeß gelegentlich in einen Zustand geraten, der ein vom normalen Verlauf abweichendes Programm erforderlich macht, muß es eine Möglichkeit geben, durch ein Sondersignal die Umschaltung auf ein Sonderprogramm (z. B. Havarieprogramm) auszulösen. Diese Einflußnahme auf den Rechner nennt man eine *Unterbrechungsanforderung* (Interruptanforderung). Die auslösende Baugruppe ist im Bild 3.21 als Prozeßkontrolleinheit (PKE) aufgeführt.

Fassen wir alle Funktionen zusammen, die ein universelles Steuersystem erfüllen muß, erhalten wir die Darstellung nach Tafel 3.8.

Wir wollen uns nun mit der Frage beschäftigen, ob alle Funktionen nach Tafel 3.8 in Form integrierter Schaltungsanordnungen, nach Möglichkeit in einem einzigen universellen Baustein, realisiert

Tafel 3.8. Übersicht über die Aufgaben eines universellen Systems zur Steuerung von Prozeßabläufen

Position Nr.	Teilaufgabe	Baugruppen zur Lösung der Aufgabe
1	Gewinnung analoger oder diskreter Meßwerte vom Prozeß	Meßfühler (Sensoren)
2	Umsetzung der analogen oder diskreten Meßwerte in digitale, codierte Meßwerte	Analog-Digital-Umsetzer (ADU)
3	Eingabesteuerung der digitalen Meßwertfolge	Eingabeeinheit (EE)
4	technisch-logische, mathematische Verarbeitung der Meßwertfolge	Digitalrechner mit folgenden Einheiten: Adressengenerator Steuerwerk Arithmetik-Logik-Einheit Registerspeicher ROM, mit Programm geladen RAM Ein-/Ausgabe-Einheiten
5	Ausgabesteuerung der digitalen Stellwertfolge	Ausgabeeinheit (AE)
6	Gewinnung diskreter, quasianaloger Stellwerte	Digital-Analog-Umsetzer
7	Reaktion auf Interruptanforderung	Prozeßkontrolleinrichtungen (PKE) Interruptaufbereitung Interruptprogramme
8	Beeinflussung des Prozesses durch Stellsignale	Stelleinrichtungen
9	Kommunikation zwischen Mensch und Automat	Eingabeeinrichtungen Ausgabeeinrichtungen

werden können. Aus der **Tafel 3.8** kann man entnehmen, daß vielfältige Aufgaben zu lösen sind, die nicht alle mit universellen Baugruppen realisiert werden können.

Zum Beispiel müssen Meßfühler eingangsseitig und Stellglieder ausgangsseitig speziell an den zu beeinflussenden Prozeß angepaßt werden. Die Palette solcher Glieder ist derart groß, daß man sie kaum universell gestalten kann, zumal nicht nur die Energieform und die physikalischen Größen, sondern auch Intensitätsbereiche und Abhängigkeiten von anderen Einflußgrößen berücksichtigt werden müssen.

Streichen wir deshalb die Teilaufgaben 1 und 8 von der Liste zur Gestaltung des universellen Steuerbausteins.

Ähnlich ist es mit den unter 7 aufgeführten Prozeßkontrolleinrichtungen, die wir ebenfalls nicht berücksichtigen.

Nehmen wir uns als nächstes die Kommunikationseinrichtungen zwischen Mensch und Automat vor. Diese Funktionsgruppen müssen an die Bedien- und Wahrnehmungsmöglichkeiten des Menschen angepaßt sein und daher notwendigerweise einen etwas größeren Raum einnehmen. Es kommen als Eingabeeinrichtungen Tastaturen, Lesegeräte, Mikrofone und als Ausgabeeinrichtungen Lampenfelder, Bildschirme, Drucker und akustische Generatoren zum Einsatz. All diese Gebilde sind einer Mikrominiaturisierung wenig zugänglich.

Verbleiben noch die Punkte 2 bis 6. Diese Funktionsgruppen eignen sich alle zur Miniaturisierung in Form mikroelektronischer Bausteine. Man hätte sich also von Anfang an mit dem Gedanken beschäftigen können, alle diese Baugruppen auf einem Chip vereinigen zu wollen. Vor der Umsetzung eines jeden Projekts muß jedoch überprüft werden, ob der Stand der Technik Erfolgschancen zur Realisierung bietet. Ein derartig umfassender Universalbaustein vom ADU über die Informationsverarbeitung im Digitalrechner bis zum DAU war bis 1979 noch nicht wirtschaftlich effektiv

herstellbar, weil zu viele Bauelemente und Baugruppen auf einem Chip vereinigt sein müssen und die Quote der nach allen Produktionsschritten funktionierenden Bausteine zu gering geworden wäre.
Deshalb wurde der Entwicklungsweg zum universellen Steuerbaustein zunächst mit einer Minimalvariante begonnen, die unter der Bezeichnung *Mikroprozessor* auf den Markt kam und nur die Funktionen der Position Nr. 4 nach Tafel 3.8 bis zu einem gewissen Teil ausführte.
Die weitere Entwicklung vollzog sich in der Weise, daß auch für die Funktionsgruppen ADU, EE, AE, DAU *Einchipvarianten* herausgebracht wurden und man den Mikroprozessor durch volle Erfüllung aller Funktionen der Nr. 4 zum *Mikrorechner* vervollkommnete. Im Jahre 1979 wurde der erste universelle Steuerbaustein auf einem Chip vorgestellt, der alle Aufgaben vom ADU bis zum DAU erfüllte.
In Zukunft wird es sicherlich weitere Fortschritte bei der Integration der in Tafel 3.8 aufgeführten Funktionsgruppen auf nur einem Chip unter Einbeziehung noch größerer Speicherkapazität geben.
Ein anderer Entwicklungstrend besteht darin, unter Anwendung der bekannten digitalen Steuersystemprinzipien hochspezialisierte Systeme, z. B. für die Sprach- und Bilderkennung oder die Sprach- und Bildgenerierung, auf einem Chip zu entwickeln.
Die Bauelementebetriebe bieten den Schaltkreisanwendern auch die Möglichkeit, mit Hilfe rechnergestützter Entwurfsverfahren ihre speziellen Schaltkreise zu entwerfen. Dabei kann der Kunde alle Entwurfsschritte bestimmen (Vollkunden-IC) oder kann vorgefertigte Elemente benutzen (Semikunden-IC).

3.7. Mikroprozessor

Die ersten integrierten Schaltkreise in Richtung des gewünschten universellen Steuerbausteins waren so ausgeführt, daß mit den damals bekannten technologischen Verfahren die Herstellungsprobleme gelöst werden konnten.
Der auf diese Weise entstandene Anfangstyp unseres Universalbausteins **(Bild 3.23)** umfaßte folgende Funktionsgruppen:

- Buspuffer
- Registerspeicher (Notizbuchspeicher (NBS), Reg. A', Reg. B', Flagregister (Flags), Kellerspeicher (STACK), Programmzähler (PC), Befehlsregister)
- Mikrosteuerwerk (MSTW)
- Arithmetik-Logik-Einheit (ALU)
- Taktgenerator (TG).

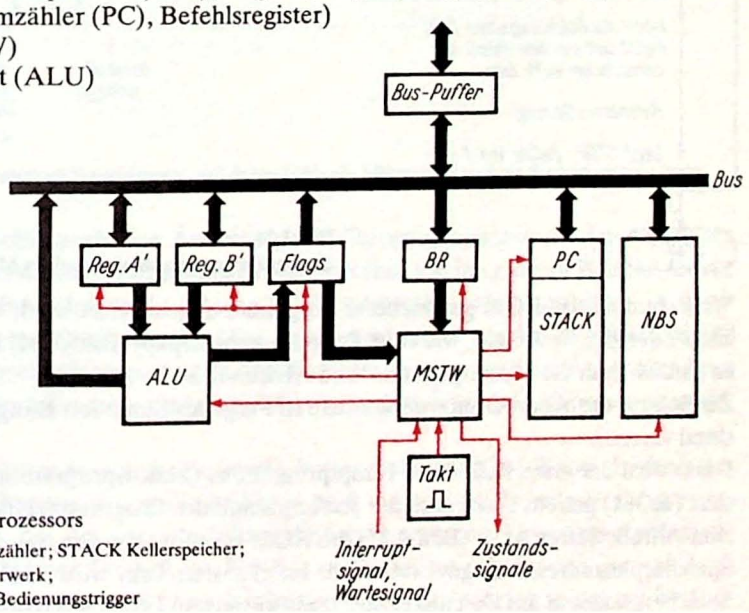


Bild 3.23. Blockbild eines Mikroprozessors
NBS Notizbuchspeicher; PC Programmzähler; STACK Kellerspeicher; BR Befehlsregister; MSTW Mikrosteuerwerk; ALU Arithmetik-Logik-Einheit; Flags Bedienungstrigger

Da dieser Baustein noch keinen Digitalrechner darstellt, wurde er *Mikroprozessor* (MP) genannt. Man kann ihn auch als Zentraleinheit (CPU = central processing unit) eines Prozeßrechners bezeichnen.
Die Zusammenschaltung aller Baugruppen erfolgt nach dem Busprinzip. Vergleicht man Bild 3.23 mit Bild 3.16, erkennt man, daß beide viele Gemeisamkeiten aufweisen. So haben beide ein Steuerwerk, einen Taktgenerator, einen Zähler, eine Addiereinrichtung, Reg. A und Reg. B. Die Ausgabeeinheit von Bild 3.16 kann man in gewisser Hinsicht mit dem Buspuffer von Bild 3.23 vergleichen, der zur Anpassung des Mikroprozessors an die externen Bausteine dient.
Damit verbleiben beim Mikroprozessor insgesamt folgende neue Funktionseinheiten, die im Bild 3.16 nicht auftreten:

- Notizbuchspeicher (NBS)
- Kellerspeicher (STACK)
- Befehlsregister (BR)
- Flagregister (Flags).

Warum hat nun der Mikroprozessor diese zusätzlichen Einheiten, die im Prinzip alle nur aus Registern bestehen?
Antwort: Wir wollen doch einen Universalbaustein haben, der mit beliebigen Steuer- und Rechenprogrammen arbeiten muß. Dazu benötigen wir einen Speicher, auf dem das Makroprogramm (Hauptprogramm) abgespeichert ist. Dieser Speicher, der als ROM gestaltet ist, wird bei großen Speicherkapazitätswerten außerhalb des Mikroprozessorchips aufgebaut (Bild 3.25). Er liefert seine Anweisungen schubweise zum Mikroprozessor, wo diese zwischengespeichert und ausgewertet werden.

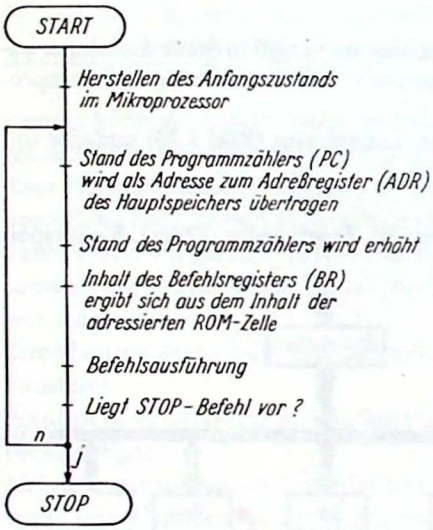


Bild 3.24
Grober Programmablaufplan eines Mikrorprozessors

Wenn nun das im ROM gespeicherte Programm abgearbeitet wird, vollzieht sich, grob gesehen, immer der gleiche Ablauf, wie er im Programmablaufplan **(Bild 3.24)** dargestellt ist. Dieser Ablauf ist ähnlich dem des Mustergenerators im Abschnitt 3.3.
Zu Beginn der Abarbeitung werden alle in Frage kommenden Baugruppen in ihren Anfangszustand versetzt.
Dann wird der erste Befehl des Hauptprogramms (Makroprogramm) vom Hauptprogrammspeicher (ROM) geholt. Dazu muß der Anfangsstand des Programmzählers (PC) über den Buspuffer zum Adreßregister ADR **(Bild 3.25)** des ROM gesendet werden, wo der Programmzählerstand als Speicherplatzadresse ausgewertet wird. Im nächsten Takt wird dann der Inhalt des adressierten Speicherplatzes in das Befehlsregister transportiert und vom Mikrosteuerwerk ausgewertet.

Dann erfolgt die Ausführung des Befehls. Ist der vom Hauptspeicher gelieferte Befehl ausgeführt worden und liegt kein STOP-Befehl vor, wird der zwischenzeitlich entsprechend erhöhte Programmzählerstand als nächste Adresse zum Adreßregister des ROM transportiert, der daraufhin den nächsten Befehl liefert. So geht es weiter, bis das gesamte Hauptprogramm abgearbeitet ist.
Das Makro-/Mikroprogramm-Prinzip hat dabei den Vorteil, daß eine Änderung des Programmablaufs nur im Hauptspeicher zu Veränderungen führt. Das Mikroprogramm im Mikrosteuerwerk bleibt davon unberührt.

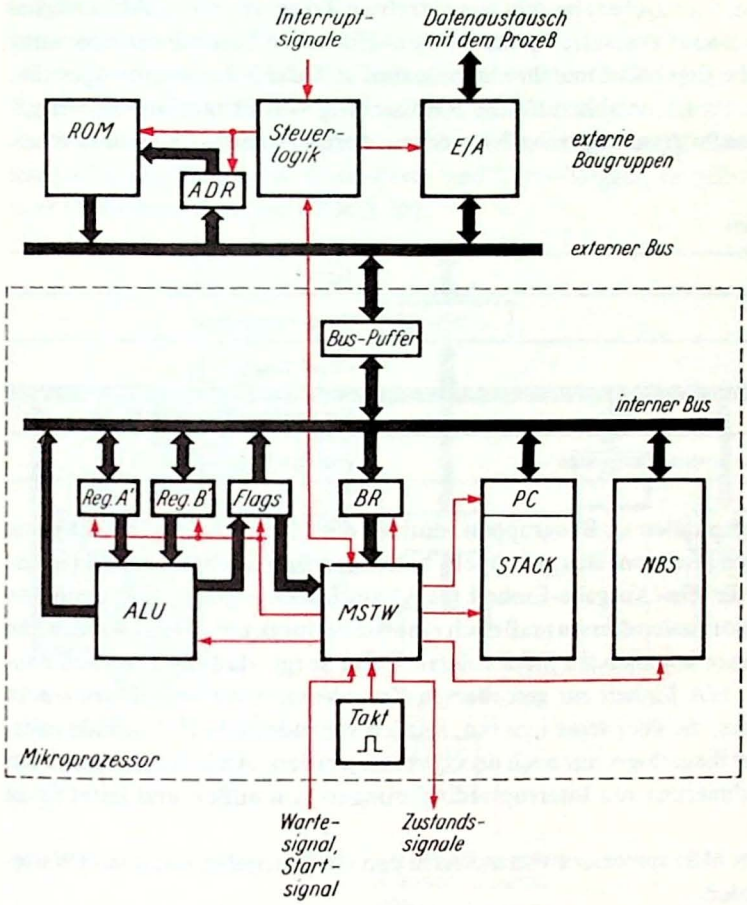


Bild 3.25. Minimalkonfiguration externer Baugruppen, mit deren Hilfe ein Mikroprozessor ein System steuern kann

Es gibt Firmen, deren Bausteine auch eine Änderung des Mikroprogramms durch den Anwender zulassen. Damit kann der Anwender spezielle Kleinsteuerwerke, wie in unserem Beispiel für die Mittelwertsbildung von Meßwertfolgen (s. Abschn. 3.5.), tatsächlich selbst gestalten. Dadurch können höhere Abarbeitungsgeschwindigkeiten des Programms erzielt werden, weil die Zeit für Adressen- und Befehlstransporte eingespart wird. Das dürfte für manche spezielle Anwendung von Wert sein. Die Gestaltung des Mikroprogramms im Mikrosteuerwerk wird als *Mikroprogrammierung* bezeichnet.
Es ist noch die Frage offen, warum man nicht die jeweils freigeschalteten ROM-Plätze gleich als Befehlsregister ausnutzt. Zur Beantwortung dieser Frage muß man den vollständigen Programmablauf betrachten und sehen, das der gesamte Informationstransport, mit Ausnahme der Steuerinformationen des MSTW, über den Bus erfolgt. Der Bus wird in jedem Takt für den Transport anderer Informationen benötigt. Daher muß er nach erfolgtem Transportvorgang sofort wieder frei-

gegeben werden, wobei man die übertragenen Zustände gleich auf der Empfangsseite einspeichern läßt. Jede Sende- und Empfangsstelle muß folglich Register oder andere Speicher aufweisen.
Der ebenfalls aus Registern bestehende Notizbuchspeicher (NBS) hat den Vorteil, daß seine Adressen sehr kurz sind, so daß geringere Speicherzugriffszeiten beim Arbeiten mit dem NBS gegenüber ROM- oder RAM-Benutzung entstehen.
Aus dem gleichen Grund des Zeitgewinns wurde der Kellerspeicher (STACK) geschaffen, dessen Register als Adressenspeicher bei Unterprogrammaufrufen verwendet werden.
Das Flagregister (Flag ≙ Fahne, Signalgeber) besteht aus einzelnen Triggern, die in Abhängigkeit vom Ergebnis der ALU gesetzt oder rückgesetzt werden und das Eintreten besonderer Ergebnisse signalisieren. Beispiele für solche Ergebnisse und ihre Signale sind in **Tafel 3.9** zusammengestellt.
Die Flagzustände werden beim Programmablauf für die Realisierung von Programmverzweigungen verwendet. Dadurch können Programme geschaffen werden, deren Ablauf sich an das Prozeßgeschehen anpaßt.

Tafel 3.9. Beispiele für Flagdefinitionen

Ergebnis der ALU-Operation	Signal
Ist 0	Zero-Flag gesetzt
Hat Übertrag	Carry-Flag gesetzt
Hat im MBS den Zustand 1	Vorzeichen-Flag gesetzt
Hat eine gerade Anzahl von Bit, die den Zustand 1 aufweisen	Paritäts-Flag gesetzt

Bild 3.25 zeigt die Minimalkonfiguration an Baugruppen, mit der ein Mikroprozessor ein System steuern kann. Über den externen (äußeren) Bus gekoppelt, beteiligen sich noch der ROM mit seinem Adreßregister ADR und die Ein-/Ausgabe-Einheit (E/A) am Datenverkehr. Zur zeitlichen Koordinierung des gesamten Informationsflusses muß noch eine Steuerlogik eingesetzt werden, die mit Hilfe der vom Mikroprozessor kommenden Steuersignale dafür sorgt, daß die externen Baugruppen ROM, Adreßregister, E/A-Einheit zur geforderten Zeit aktiviert werden. Dazu sendet das Mikrosteuerwerk Signale aus, die über seine inneren, zeitlich veränderlichen Zustände informieren. Diese brauchen von der Steuerlogik nur noch decodiert zu werden. Außerdem übernimmt die Steuerlogik auch die Koordinierung von Interruptanforderungen von außen und leitet sie an das Mikrosteuerwerk weiter.
Durch das Wartesignal kann der Mikroprozessor von außen in den vorübergehenden Halt (Warte- [Wait-] Zustand) gesteuert werden.

3.8. Einfacher Mikroprozessor

Wir wollen uns nun mit Entwurfsüberlegungen zum Signalflußbild eines einfachen Mikroprozessors befassen. Sie werden sicherlich ahnen, daß wir nicht zu tief in die Detailbetrachtungen gehen können, weil das die Darstellungsmöglichkeiten in diesem Buch übersteigt. Doch wollen wir wiederum so viele Einzelheiten aufnehmen, daß ein tiefergehendes Verständnis für die Wirkungsweise dieses Mikroprozessors entsteht.
Wir nehmen uns absichtlich keinen gefertigten Typ vor, sondern projektieren einen, der uns beim Gewinnen von Einsichten bestmöglich helfen soll.
Er soll 16 verschiedene elementare Befehle, darunter Transportbefehle, arithmetische Verknüpfungen, Programmverzweigungsbefehle, ausführen können. Die Anzahl von 16 Befehlen stellt einen Kompromiß im Interesse der besseren Übersicht dar. Mit einer größeren Anzahl von Befeh-

len kann man bequemer programmieren, die geringe Anzahl von 16 Befehlen ist beim Lernenden leichter einprägbar.
Unser Mikroprozessor soll einen Bus aus acht parallelen Leitungen bekommen. Daten- und Adreßwort sollen je 8 Bit = 1 Byte breit sein, damit der Bus voll ausgenutzt wird. Mit 8 Bit Wortbreite lassen sich dann jeweils 256 verschiedene Wörter darstellen.
Eine Adreßwortbreite von 8 Bit bedeutet, daß ein externer Speicher mit 256 Speicherwörtern adressierbar ist. Das heißt z. B., daß 256 Befehle in ihrer gespeicherten Reihenfolge aufgerufen werden können.
Unser Mikroprozessor soll drei als Notizbuchspeicher wirkende Register (Reg. A, Reg. B, Reg. C) erhalten, von denen Reg. A als Akkumulator dient. Gleichzeitig bilden Reg. A und Reg. B auch die Operandenspeicher für die Arithmetik-Logik-Einheit (ALU).
Außerdem soll er einen Programmzähler (PC), einen STACK mit einem Register, ein Auffangregister (AR), ein Flagregister mit Zero- und Carry-Trigger, Befehlsregister (BR) und Mikrosteuerwerk (MSTW) aufweisen **(Bild 3.26)**.

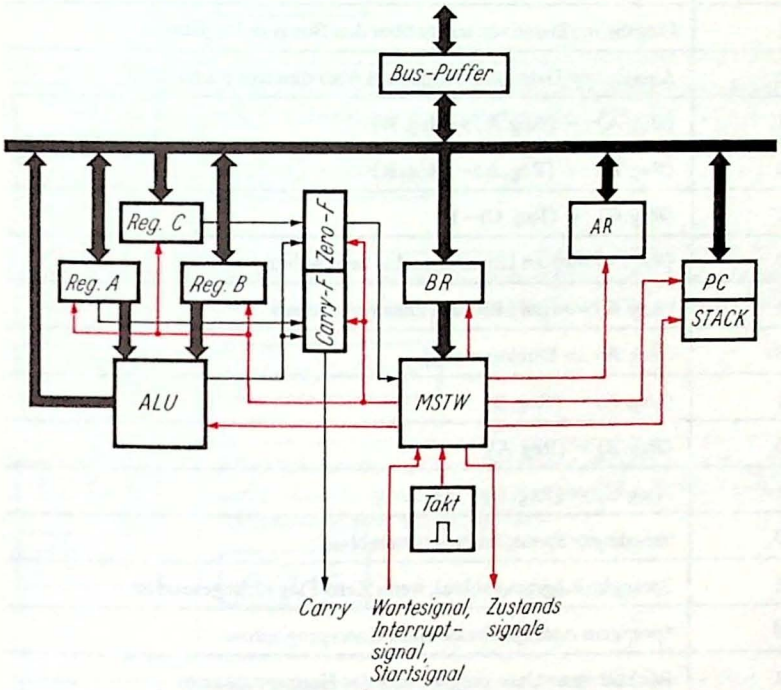


Bild 3.26. Blockbild eines einfachen Mikroprozessors

Das Flagregister hat eine Breite von 2 Bit; das Befehlsregister ist 4 Bit, alle anderen Register sind 8 Bit breit.
Er soll auf Warte- und Interruptsignal reagieren und die Zustände des Mikrosteuerwerks nach außen melden können.

3.8.1. Befehlsliste

Als nächste Maßnahme geben wir die 16 Befehle vor. Diese sind in **Tafel 3.10** zusammengestellt. Sie sind so ausgesucht worden, daß man trotz der geringen Befehlsanzahl schon Programme realisieren kann. Sie stellen nach Art und Bezeichnung eine Auswahl der wichtigsten Befehle real existierender Mikroprozessoren dar.

Die Tafel 3.10 hat neben der Numerierungsspalte drei Aussagespalten. Von diesen trägt die erste Spalte als Überschrift das schwierig aussprechbare Wort *Mnemonik*. Nach dem Duden bedeutet es die Kunst, das Gedächtnis durch Hilfsmittel (Merkworte, Merksätze) zu unterstützen. In unserem Fall stellt die Mnemonik Merkwörter für die einzelnen Befehle dar.
Die gebräuchlichen Mnemonikausdrücke entstammen alle dem anglo-amerikanischen Sprachbereich und bieten für Unkundige dieser Sprache größtenteils wenig merktechnische Erleichterung, müssen also leider erlernt werden.
In der nächsten Spalte sind die einzelnen Befehle in ihrem Binärcode dargestellt. Da es 16 Befehle sind, wurden sie in der Folge der Dualzahlen von null bis fünfzehn aufgeführt.
Die letzte Spalte beschreibt in Kurzform die Wirkung der einzelnen Befehle.

Tafel 3.10. Definitionstabelle der 16 Befehle für den einfachen Mikroprozessor

Nr.	Mnemonik	Binärcode	Wirkung
1	HLT	0 0 0 0	STOP der Abarbeitung des Mikroprogramms
2	IN	0 0 0 1	Eingabe von Daten von außen über den Bus zum Register A
3	OUT	0 0 1 0	Ausgabe von Daten vom Register A über den Bus nach außen
4	ADD B	0 0 1 1	⟨Reg. A⟩ := ⟨Reg. A⟩+⟨Reg. B⟩
5	SUB B	0 1 0 0	⟨Reg. A⟩ := ⟨Reg. A⟩−⟨Reg. B⟩
6	DCR C	0 1 0 1	⟨Reg. C⟩ := ⟨Reg. C⟩−1
7	RRC	0 1 1 0	⟨Reg. A⟩ wird um 1Bit nach rechts verschoben
8	RLC	0 1 1 1	⟨Reg. A⟩ wird um 1 Bit nach links verschoben
9	MVI A	1 0 0 0	⟨Reg. A⟩ := Direktoperand
10	MOV A,B	1 0 0 1	⟨Reg. A⟩ = ⟨Reg. B⟩
11	MOV B,A	1 0 1 0	⟨Reg. B⟩ = ⟨Reg. A⟩
12	MOV C,A	1 0 1 1	⟨Reg. C⟩ = ⟨Reg. A⟩
13	JMP	1 1 0 0	unbedingter Sprung im Programmablauf
14	JNZ	1 1 0 1	Sprung im Programmablauf, wenn Zero-Flag nicht gesetzt ist
15	CALL	1 1 1 0	Sprung zur Anfangsadresse eines Unterprogramms
16	RET	1 1 1 1	Rückkehr vom Unterprogramm in das Hauptprogramm
		$a_3a_2a_1a_0$	

Sehen wir uns als nächstes **Tafel 3.11** an, in der alle Befehle ausführlich erläutert werden. Zur Verkürzung der Schreibweise wollen wir noch vereinbaren, daß anstelle der Winkelklammerausdrücke für Registerinhaltsangaben auch kleine Buchstaben verwendet werden sollen, z. B.

- $\langle \text{Reg.A} \rangle = a$
  $\langle \text{Reg.B} \rangle = b$ usw.

Damit haben wir eine ähnliche Vereinbarung wie auf S. 40 getroffen, nach der Schaltungspunkte oder Klemmen mit großen Buchstaben und die an den Klemmen auftretenden Variablen mit den gleichen Kleinbuchstaben bezeichnet wurden. Analog werden jetzt die Register mit großen und ihr Inhalt mit kleinen Buchstaben dargestellt.

Nach ihrer Wirkung können die Befehle unseres Mikroprozessors in drei große Gruppen zusammengefaßt werden:

- Übertragungsbefehle
- Befehle für arithmetische Operationen
- Befehle zur Programmorganisation.

Übertragungsbefehle. Hierzu gehören

IN, OUT, MVI A, MOV A,B, MOV B,A, MOV C,A.

Tafel 3.11. Ausführliche Erläuterung der 16 Befehle des einfachen Mikroprozessors

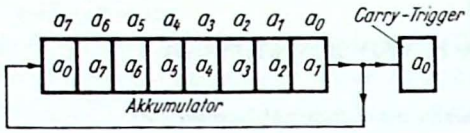
HLT 0000	Der Mikroprozessor geht in den STOP-Zustand. Er führt keine Befehle mehr aus. Aus dem STOP-Zustand kann er nur herauskommen, wenn er von außen wieder gestartet wird (Startimpulsgabe) STOP
IN 0001	IN ≙ EIN. Der Mikroprozessor wertet die am Bus von außen gelegten Daten als Eingabedaten und holt sich eine Kopie dieser Werte in den Akkumulator Reg. A ⟨Reg. A⟩ := ⟨ext. Reg.⟩
OUT 0010	OUT ≙ AUS. Der Mikroprozessor gibt den Inhalt des Akkumulators Reg. A auf den Bus, so daß diese Daten von einem externen Register übernommen werden können ⟨ext. Reg.⟩ := ⟨Reg. A⟩
ADD B 0011	ADD kommt von Addition. Die Summe der Inhalte von Reg. A und Reg. B wird durch Aktivierung der Tristate-Ausgangsorte der ALU (im Bild 3.26 nicht im Detail dargestellt) über den Bus in den Akkumulator Reg. A eingespeichert $a := a + b$
SUB B 0100	SUB kommt von Subtraktion. Es wird die Summe vom Inhalt des Reg. A mit dem Zweierkomplement des Inhalts von Reg. B gebildet. Die damit entstandene Differenz der beiden Registerinhalte wird im Akkumulator Reg. A abgespeichert $a := a - b$
DCR C 0101	DCR leitet sich von Dekrement (stufenweise Verringerung) ab. Reg. C, das als Rückwärtszähler arbeitet, erhält durch diesen Befehl einen Zählimpuls. Dadurch vermindert sich der Registerinhalt um eins. Das Zero-Flag wird jeweils nach Erreichen eines neuen Zählerstands z beeinflußt. Setzimpuls bei $z = 0$; Rücksetzimpuls bei $z \neq 0$ $c := c - 1$

Tafel 3.11. (Fortsetzung)

RRC

0 1 1 0

RRC ist abgeleitet vom englischen Text: „rotation right and shift into carry."
Das bedeutet: „Rotation rechts herum und Verschiebung in den Übertrag."
Beim Aufruf dieses Befehls wird der Akkumulatorinhalt um eine Stelle nach rechts verschoben. Das aus dem Reg. A austretende Bit wird im Carry-Trigger gespeichert und gleichzeitig am anderen Ende des Akkumulators wieder eingegeben.



Zustand nach einmaliger Ausführung des Befehls RRC

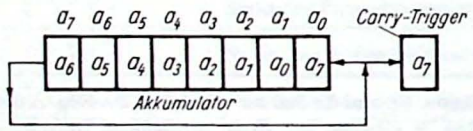
	⟨Reg. A⟩								
n	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Carry
0	1	0	0	1	1	1	0	1	–
1	1	1	0	0	1	1	1	0	1
2	0	1	1	0	0	1	1	1	0
3	1	0	1	1	0	0	1	1	1
4	1	1	0	1	1	0	0	1	1
5	1	1	1	0	1	1	0	0	1
6	0	1	1	1	0	1	1	0	0
7	0	0	1	1	1	0	1	1	0
8	1	0	0	1	1	1	0	1	1

Beispiel für mehrmalige (8-malige) Ausführung des Befehls RRC
Für n=0 wurde der dargestellte Anfangszustand beliebig ausgewählt

RLC

0 1 1 1

RLC leitet sich von „rotation left and shift into carry" (Rotation links herum und Verschiebung in den Übertrag) ab.
Der Akkumulatorinhalt wird um eine Stelle nach links verschoben. Das aus dem Akkumulator austretende Bit wird im Carry-Trigger gespeichert und gleichzeitig am anderen Ende des Akkumulators wieder eingegeben.



Zustand nach einmaliger Ausführung des Befehls RLC

MVI A

1 0 0 0

$d_3 d_2 d_1 d_0$

$d_7 d_6 d_5 d_4$

MVI A kommt von „move immediate into register A" (gib direkt weiter in Reg. A).
Diejenige Zahl, die direkt weitergegeben werden soll, wird unmittelbar nach dem Befehlswort 1000 auf den nächsten beiden ROM-Plätzen gespeichert. Diese Zahl wird als Direktoperand bezeichnet. Nach Decodierung des Befehls MVI A werden die Inhalte der nachfolgenden beiden ROM-Plätze als Teile eines Direktoperanden gewertet und zum Auffangregister übertragen. Zunächst wird der niederwertige Teil (d_3 d_2 d_1 d_0) und dann der höherwertige Teil (d_7 d_6 d_5 d_4) vom Auffangregister übernommen. Danach wird eine Kopie des gesamten Inhalts vom Auffangregister zum Akkumulator übertragen

$a := \text{Direktoperand} = d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0$

Tafel 3.11. (Fortsetzung)

MOV A,B 1 0 0 1	MOV kommt vom englischen „move" (bewegen, weitergeben). MOV A,B heißt soviel wie „Gib den Inhalt des Registers B an den Akkumulator weiter" $a := b$
MOV B,A 1 0 1 0	Der Inhalt des Akkummulators wird zum Reg. B übertragen $b := a$
MOV C,A 1 0 1 1	Der Inhalt des Akkummulators wird zum Reg. C übertragen $c := a$
JMP 1 1 0 0 $z_3\ z_2\ z_1\ z_0$ $z_7\ z_6\ z_5\ z_4$	JMP kommt von „jump" (Sprung). Im Programmablauf wird ein unbedingter Sprung bezüglich der weiteren Befehlsreihenfolge ausgeführt. Auf den beiden nach der Befehlscodierung 1100 folgenden ROM-Plätzen steht die Adresse des als nächsten abzuarbeitenden Befehls. Zuerst wird der niederwertige Teil der Adresse, dann der höherwertige Teil in das Auffangregister übertragen. Mit der vollständigen Adresse wird dann der neue Stand des Programmzählers gebildet $\langle PC \rangle := z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0$
JNZ 1 1 0 1 $z_3\ z_2\ z_1\ z_0$ $z_7\ z_6\ z_5\ z_4$	JNZ kommt von „jump if not zero" (Springe nur, wenn das Zero-Flag nicht gesetzt ist). Sprungbefehl, der nur dann ausgeführt wird, wenn das Zero-Flag den Zustand 0 einnimmt. Wegen der Abhängigkeit der Befehlsausführung von einer Bedingung wird dieser Sprungbefehl als bedingter Sprungbefehl bezeichnet. Der Inhalt der beiden folgenden ROM-Plätze liefert die Adresse des als nächsten abzuarbeitenden Befehls. Das Auffangregister übernimmt schubweise die gesamte Adresse und lädt damit anschließend den Programmzähler $\langle PC \rangle := z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0$
CALL 1 1 1 0 $z_3\ z_2\ z_1\ z_0$ $z_7\ z_6\ z_5\ z_4$	CALL bedeutet Ruf: Aufruf eines Unterprogramms. Unbedingter Sprung zur Anfangsadresse eines Unterprogrammteils. Diese Adresse wird von den folgenden beiden ROM-Plätzen über das Auffangregister als neuer Stand zum Programmzähler übertragen. Der alte Programmzählerstand wird nach der Erhöhung im STACK gespeichert und kann nach Abarbeitung des Unterprogramms wieder als Fortsetzungsadresse verwendet werden $\langle STACK \rangle := \langle PC \rangle$ $\langle PC \rangle := z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0$

Tafel 3.11. (Fortsetzung)

RET 1 1 1 1	RET kommt von „return" (Rückkehr). Rückkehrbefehl. Befehl zur Wiederverwendung des im STACK abgelegten alten Programmzählerstandes. Mit der im STACK gespeicherten Adresse wird der Programmzähler geladen. Damit wird nach zwischenzeitlicher Abarbeitung eines Unterprogramms die Abarbeitung des Hauptprogramms fortgesetzt. Mit dem Befehl RET muß ein mit CALL aufgerufenes Unterprogramm abgeschlossen werden ⟨PC⟩:=⟨STACK⟩

Unter *Übertragung* verstehen wir die parallele Weitergabe des Inhalts eines Registers (Speichers) in ein anderes Register. Hierbei bleibt der Inhalt des Absenderregisters erhalten. Bei der Übertragung wird praktisch nur eine Kopie des Inhalts vom Absenderregister übermittelt.
Will man ein Register mit Hilfe der Befehlsgabe im Programm löschen, muß in dieses Register die Dualzahl 0000 0000 eingeschrieben werden. Diese Rechenzeit in Anspruch nehmende Maßnahme wird natürlich nur dann angewendet, wenn sie unbedingt erforderlich ist. Liegt kein Grund zur Löschung eines Registers vor, so läßt man seinen Inhalt einfach stehen.
Wie Sie sicherlich gemerkt haben, wurden auf Grund der auf 16 begrenzten Befehlsanzahl nicht alle möglichen Übertragungsvarianten zwischen den drei Registern des Notizbuchspeichers realisiert. Welche Möglichkeiten fehlen? Bitte überlegen Sie selbst!
Antwort: Es fehlen MOV A,C, MOV B,C, MOV C,B.
Befehle für arithmetische Operationen. Hierzu gehören

ADD B, SUB B, DCR C, RRC, RLC.

Die meisten arithmetischen Befehle beziehen den Akkumulatorinhalt ein. So hat ADD B die Wirkung ⟨Reg. A⟩ := ⟨Reg. A⟩ + ⟨Reg B⟩, und SUB B führt zu ⟨Reg. A⟩ := ⟨Reg. A⟩ − ⟨Reg. B⟩. Bei RRC und RLC werden Verschiebungen im Akkumulator ausgeführt. Damit kann man Multiplikation und Division realisieren. Außerdem ermöglichen diese Befehle eine zyklische Vertauschung der Bitstellen.
Nur beim Befehl DCR C ist der Akkumulator nicht einbezogen.
Befehle zur Programmorganisation. Hierzu gehören

HLT, JMP, JNZ, CALL, RET.

Bis auf den Haltbefehl führen alle anderen zu Programmverzweigungen, d. h. zu Fortsetzungen der Befehlsabarbeitung an einer anderen Stelle als derjenigen, die beim letzten, vorangegangenen Befehl durch den Programmzählerstand vorausgebildet wurde. Der CALL-Befehl führt zur Aufbewahrung des gerade gebildeten Programmzählerstands im STACK und zur Abarbeitung einer Unterprogrammbefehlsfolge (s. auch S. 189), nach deren Beendigung der Befehl RET folgen muß, wodurch der im STACK aufbewahrte Zählerstand wieder in den Programmzähler geladen wird. Anschließend wird das Programm mit dem Befehl fortgesetzt, dessen Adresse dem gerade geladenen Programmzählerstand entspricht.
Durch die Befehle CALL und RET wird also ein Stück Befehlsfolge zur Abarbeitung eingeschoben.
In Realisierung eines Sprungbefehls wird die Programmabarbeitung an einer anderen Stelle fortgesetzt, ohne daß eine Rückkehradresse im STACK gespeichert wird. Der Sprungbefehl soll also vorzugsweise zur Änderung der Abarbeitsfolge ohne Rückkehrabsicht dienen.
Natürlich lassen sich auch Unterprogramme mit dem Sprungbefehl anspringen. Dann muß aber nach Beendigung des Unterprogramms wieder ein Sprungbefehl mit neuer Adressenangabe zur Rückkehr in das Hauptprogramm gegeben werden.

3.8.2. Entwurfsüberlegungen zu Einzelheiten des Blockbilds

Die Wirkungsweise der einzelnen Blöcke des Blockbilds (Bild 3.26) unseres Mikroprozessors ist uns aus vorhergehenden Abschnitten im wesentlichen bekannt. Wir können uns daher entsprechend kurz fassen.

Beginnen wir mit dem Notizbuchspeicher. Das Register A dient als Akkumulator und muß rechts/links schieben können. Zusätzlich muß es, wie Register B, parallele Daten empfangen und senden können. Wegen des Befehls DCR C soll Register C als Rückwärtszähler ausgebildet sein.

Gehen wir weiter zum Programmzähler. Er ist ein Vorwärtszähler, der zur Bildung der ROM-Adressen in Form des Dualzahlenmusters dient. Zu Beginn der Abarbeitung eines jeden Befehls wird der Stand des Programmzählers zum Adreßregister des ROM übertragen und adressiert die Speicherzelle, die diesen nächsten Befehl enthält. Gleich anschließend empfängt der Programmzähler einen Impuls zur Erhöhung seines Zählerstands. Damit liegt zu Beginn der nächsten Befehlsabarbeitung die neue Adresse bereits vor.

Die Adressenbildung vollzieht sich also durch gleichmäßige Aufwärtszählung im Dualzahlenschema, wenn nicht ein Sprungbefehl oder ein Unterprogrammaufruf die kontinuierliche Reihenfolge unterbricht.

Gehen wir weiter zum Buspuffer. Er stellt die Anpassung des Mikroprozessors an die externe Schaltung bezüglich Leistungs- und Spannungspegel sicher. Da er von speziellen Ausführungsvarianten abhängig ist, soll er hier nicht weiter besprochen werden.

Betrachten wir nun den Komplex ALU, Reg. A, Reg. B, Reg. C, Flagregister. In den Registern Reg. A und Reg. B werden die Daten gespeichert, die von der ALU verknüpft werden sollen. Das Ergebnis der ALU-Operation wird zum Akkumulator Reg. A übertragen. Tritt im Ergebnis der ALU-Operation oder beim Zählvorgang im Register C die Zahl null auf, so wird der Zero-Trigger (Zero-Flag) in den Zustand 1 und bei allen anderen Ergebnissen in den Zustand 0 gesetzt. Tritt im Ergebnis der ALU-Operation ein Übertrag auf, wird der Carry-Trigger (Carry-Flag) in den Zustand 1 und bei fehlendem Übertrag in den Zustand 0 gesetzt.

Verbleiben das Befehlsregister und das Mikrosteuerwerk. Das Befehlsregister dient zur Übernahme der vom ROM geholten Befehle. Sein Inhalt wirkt auf das Mikrosteuerwerk als Steuerbedingung ein.

Anschließend wollen wir das Kernstück des Mikroprozessors, das Mikrosteuerwerk, in einem eigenen Abschnitt behandeln.

3.8.3. Entwurfsüberlegungen zum Mikrosteuerwerk

Aus dem Abschnitt 3.5 (Bilder 3.19 und 3.20) wissen wir bereits, wie ein Mikrosteuerwerk aussieht. Dieses dort gezeigte Steuerwerk wird an unser Problem angepaßt.

Die Interruptgabe soll sehr einfach funktionieren und nur Start und außerplanmäßigen STOP realisieren. Bei Signalgabe an WAIT soll der Wartezustand eingenommen werden. Zugeführt werden muß noch der Takt, und das Steuerwerk soll seinen nach jedem Taktimpuls jeweils eingenommenen Arbeitszustand nach außen melden. Die vom ROM abgerufenen Befehle werden als Bedingungen gewertet.

Bevor wir das Mikrosteuerwerk im einzelnen betrachten, wollen wir noch einige Überlegungen zum vorgegebenen Befehlsformat kennenlernen.

Nach Tafel 3.10 ist für jeden Befehl ein 4 Bit breiter Binärcode vorgesehen. Das ist die Grundinformation, mit der 75 % der Befehle hinreichend definiert sind. Die anderen Befehle benötigen neben den ausgewiesenen 4 Bit noch zusätzliche 8 Bit zur Darstellung einer Zahl (MVI A) oder einer Adresse (JMP, JNZ, CALL), müßten also aus 12 Bit breiten Wörtern bestehen.

Nun ist es aber technisch am einfachsten, bei der Festlegung der Speicherstruktur des ROM nur von gleich breiten Wörtern auszugehen, da man von vornherein nicht weiß, welcher Befehlstyp unter welche Adresse zu stehen kommt. Deshalb müssen wir uns hier entscheiden, ob wir eine 4-Bit- oder 12-Bit-Speicherstruktur festlegen.

Nehmen wir eine 12-Bit-Speicherwortbreite, so bleiben immer dann 8 Bit frei, wenn wir einen Befehl verwenden, der außer seinem Befehlscode nichts weiter mitzuteilen hat. Das ist bei 75 % der Befehle der Fall, die noch dazu häufiger als die anderen 25 % der Befehle verwendet werden. Außerdem müßte der Bus zur gleichzeitigen Übertragung dieser 12 Bit auch aus 12 Leitungen bestehen.

Zur effektiven Speicherausnutzung soll deshalb eine 4-Bit-Speicherwortbreite gewählt werden. Das bedeutet, daß die 12 Bit breiten Befehlswörter in drei Schüben (Zyklen) übertragen werden müssen. Dies erfordert mehr Verarbeitungszeit. Was wir also an Effektivität bei der Ausnutzung des Speichers und des Busses gewinnen, müssen wir an Abarbeitungszeit zugeben.

Diese Entscheidung zu 4 Bit breiten Wörtern für die ROM-Organisation hat entsprechende Auswirkungen auf das Mikrosteuerwerk. So wird dann z. B. ein vollständiger Befehl MVI A wie folgt aussehen:

1. Speicherwort 1 0 0 0 Befehlscode
2. Speicherwort d_3 d_2 d_1 d_0 niederwertiger Teil des Direktoperanden
3. Speicherwort d_7 d_6 d_5 d_4 höherwertiger Teil des Direktoperanden

Die drei Speicherwörter müssen im Speicher in dieser Reihenfolge zusammenbleiben. Hat das 1. Speicherwort die Adresse n, so muß die Adresse des 2. Speicherworts $n + 1$ und die des dritten $n + 2$ lauten.

Die Abarbeitung des Befehls MVI A vollzieht sich dann in folgenden Schritten:

Schritt 1. Lesen des ROM-Speicherplatzes mit der Adresse n. Das Mikrosteuerwerk *erkennt* dann, daß es sich um einen Direktoperandentransport handelt, und weiß, daß der Direktoperand mit dem Inhalt der nächsten beiden ROM-Speicherplätze übertragen wird.

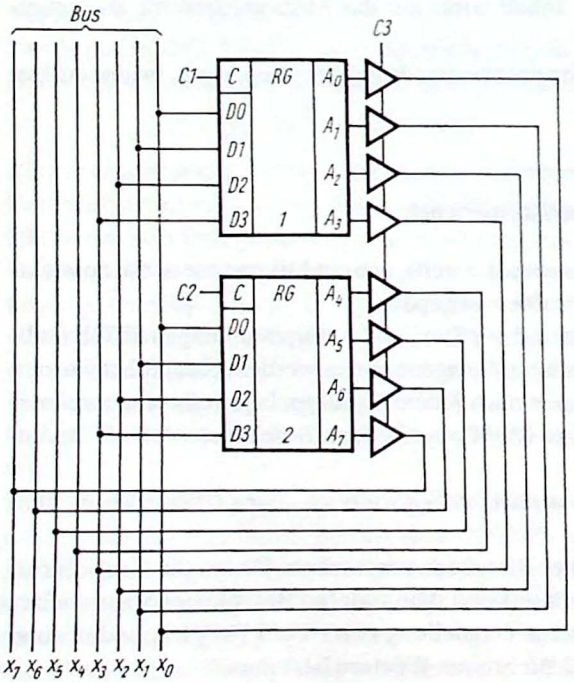


Bild 3.27
Signalflußbild des Auffangregisters

Schritt 2. Lesen des ROM-Speicherplatzes mit der Adresse $n + 1$. Das Mikrosteuerwerk muß veranlassen, daß der nun vom ROM gelieferte niederwertige Teil des Direktoperanden im niederwertigen Teil des Auffangregisters (Reg. 1 im **Bild 3.27**) gespeichert wird.
Durch die Übertragung eines Übernahmeimpulses nach C1 wird die Information in das Auffangregister eingespeichert und steht dann an den Ausgängen A0 bis A3 zur Verfügung.
Schritt 3. Lesen des ROM-Speicherplatzes mit der Adresse $n + 2$. Der nun vom ROM gelieferte höherwertige Teil des Direktoperanden wird über die Busleitungen X0 bis X3 zu den Eingängen D0 bis D3 des höherwertigen Teils des Auffangregisters (Reg. 2) geleitet und durch Übertragung eines Übernahmeimpulses nach C2 eingespeichert, so daß nun der gesamte Direktoperand an den Ausgängen A0 bis A7 zur Verfügung steht.
Abschließend wird der an A0 bis A7 auftretende Direktoperand über den Bus zum Akkumulator übertragen. Dazu müssen die Ausgänge des Auffangregisters über Tristate-Gatter mit dem Bus verbunden werden. An C3 wird das Toröffnungssignal gelegt.
Zum besseren Verständnis sei noch ein Zahlenbeispiel mit **Tafel 3.12** angefügt.
Ebenso muß auch ein Sprungbefehl aus drei Speicherwörtern bestehen. Die Speicherwortfolge

1 1 0 0	Befehlscode
0 0 1 1	niederwertiger } Teil des Direktoperanden
1 1 1 1	höherwertiger }

bedeutet dann:
Es ist ein unbedingter Sprung zur ROM-Adresse 11110011 auszuführen. Auch hierbei werden die beiden Teile des Direktoperanden im Auffangregister zur vollständigen 8-Bit-Information zusammengesetzt. Im Unterschied zum Befehl MVI A wird aber der Inhalt des Auffangregisters anschließend nicht zum Akkumulator übertragen, sondern dient zum Laden des Programmzählers.
Nach diesen ersten Überlegungen wollen wir uns nun mit Start und STOP der Programmabarbeitung befassen.
Die Anlaufphase des Steuerwerks soll sich so vollziehen, daß nach Anlegen der Versorgungsspannung durch eine Anlaufsteuerschaltung sämtliche in Frage kommenden Trigger, Register, Zähler rückgesetzt werden. Der reguläre Start soll dann durch Auslösen eines Startimpulses von Hand erfolgen.
Den Anlaufprozeß wollen wir schaltungstechnisch nicht betrachten. Bei unserem Entwurf beginnen wir beim Schaltzustandstrigger. Dieser soll als Einschalter beim Start und als Ausschalter bei Interruptsignal wirken. Wir benutzen dazu einen RS-Trigger.
Bei STOP durch Interruptsignal soll der gerade in Bearbeitung befindliche Befehl noch abgearbeitet werden, und dann soll das Mikrosteuerwerk an einer Stelle ohne Operationsdurchführung im ordnungsgemäßen STOP-Zustand, der auch durch den Befehl 0000 erreicht wird, verharren.

Tafel 3.12. Erläuterung des Ablaufs eines Direktoperandentransports vom Programmspeicher in den Akkumulator (Wirkung des Befehls MVI A)

Schritt	Aufgerufene ROM-Adresse	Inhalt des adressierten ROM-Platzes	Bedeutung des ROM-Inhalts	Inhalt des Auffangregisters nach dem Schritt	Inhalt des Akkumulators nach dem Schritt
1	n	1 0 0 0	Code für den Befehl MVI A	0000 0000	0000 0000
2	$n + 1$	1 1 0 0	niederwertiger Teil des Direktoperanden	0000 1100	0000 0000
3	$n + 2$	0 1 0 1	höherwertiger Teil des Direktoperanden	0101 1100	0101 1100

Den groben Programmablaufplan des Mikrosteuerwerks zeigt **Bild 3.28**.
Nach dem automatischen Herstellen des Anfangszustands befindet sich das Mikrosteuerwerk im Zustand, den die Schleife 1 des Programmablaufplans beschreibt. Wir werden später im Signalflußbild des Mikrosteuerwerks sehen, daß in dieser Schleife bei jeder Taktimpulsgabe überprüft wird, ob der STOP nicht schon durch Gabe des Startimpulses aufgehoben ist.
Ist dies der Fall, so wird als nächste Operation der Stand des Programmzählers als Adresse zum Adreßregister des ROM übertragen. Liegt eine Warteanforderung vor (WAIT-Signal im Zustand 1), wird anschließend der weitere Ablauf durch die WAIT-Schleife (Schleife 2) bestimmt. Auch in dieser Schleife wird bei jeder Taktimpulsgabe überprüft, ob das WAIT-Signal schon den Zustand 0 angenommen hat. Ist dies der Fall, so wird der Befehl in das Befehlsregister übertragen und decodiert. Danach wird der Befehl ausgeführt.

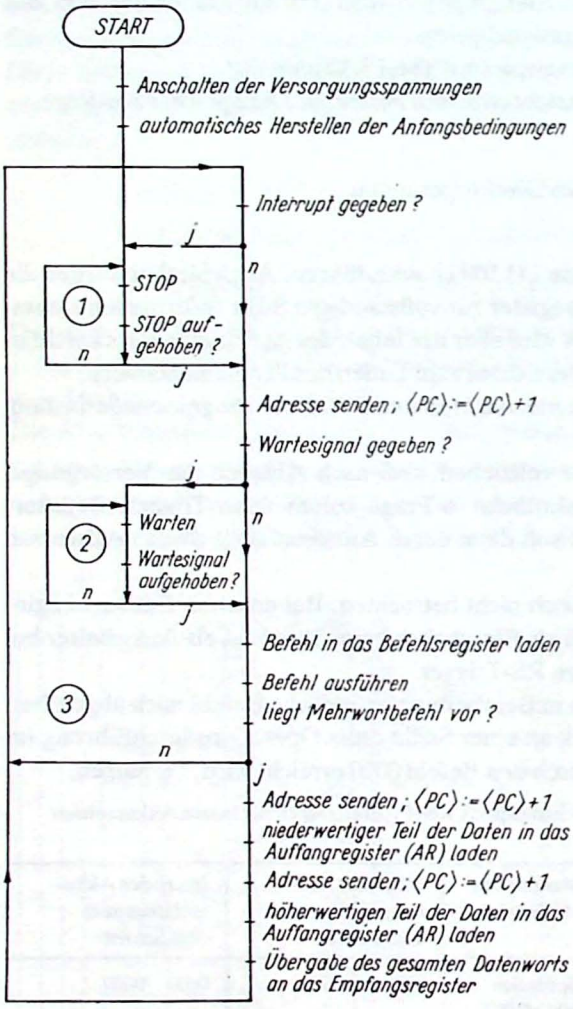


Bild 3.28
Grober Programmablaufplan des einfachen Mikroprozessors

Bei einem Drei-Wort-Befehl werden anschließend die beiden anderen Speicherplätze des ROM gelesen, deren Inhalt im Auffangregister gesammelt und dann zum Empfangsregister übertragen wird.
Liegt nur Ein-Wort-Befehl vor, wird der Abarbeitungsumlauf über die Schleife 3 geschlossen. Der neue Abarbeitungsumlauf beginnt dann mit der Frage, ob inzwischen ein Interruptsignal gegeben worden ist.

In dieser Form wird ständig der Programmablaufplan des Mikrosteuerwerks abgearbeitet, bis entweder ein Haltbefehl erscheint oder Interruptsignal gegeben wird.

Schauen wir uns nach dieser Vorbereitung die ***Bilder 3.29 und 3.30*** *sowie* ***Tafel 3.13*** *an, die uns die Funktion der wesentlichen Teile des Mikrosteuerwerks erkennen lassen sollen.*

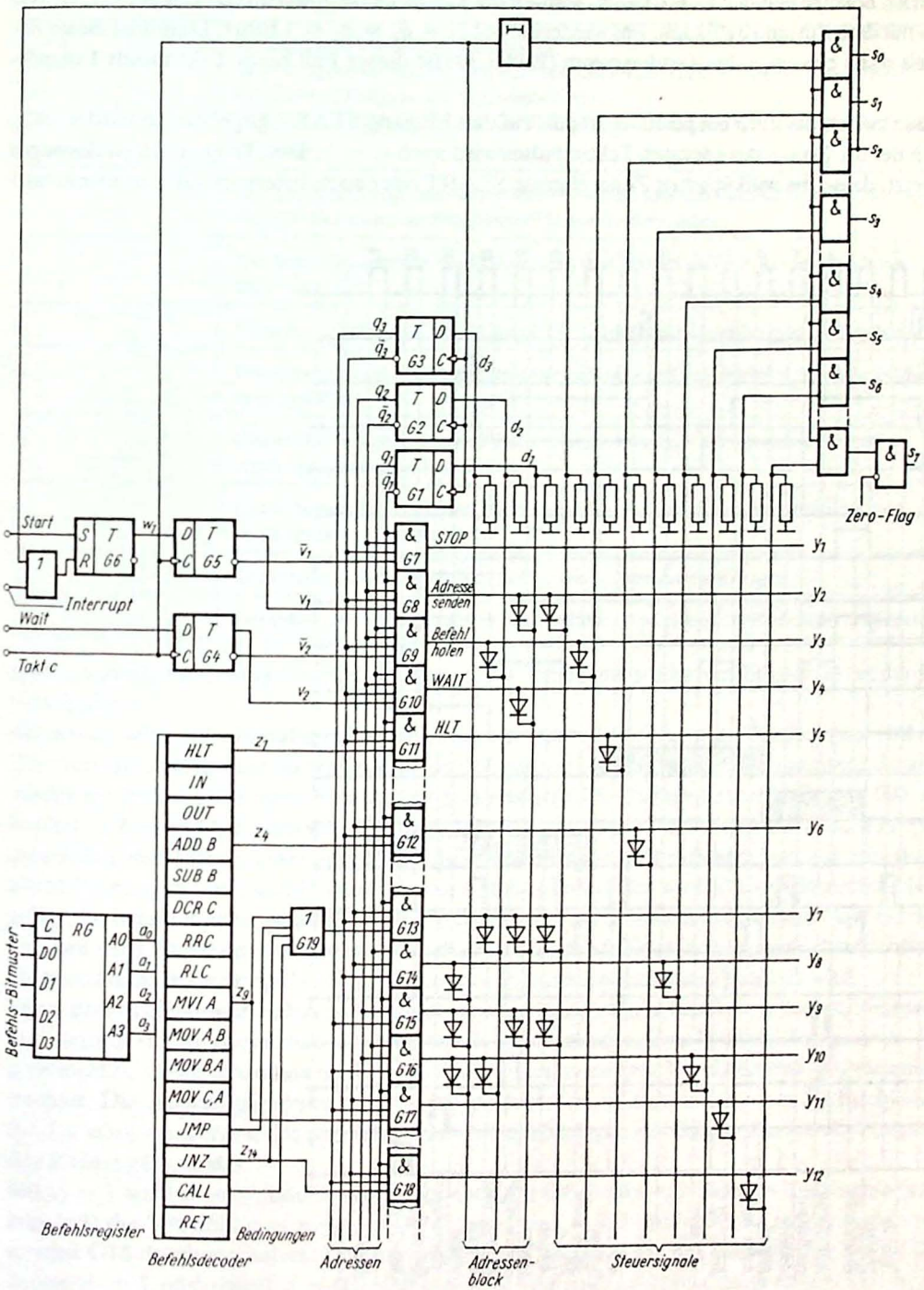


Bild 3.29. Auszugsweise Darstellung eines programmierten Mikrosteuerwerks für den einfachen Mikroprozessor

Wir beginnen mit der STOP-Schleife (Zeile y_1). Als Adresse $d = d_3d_2d_1$ liegt hier der Wert $d = 000$ vor. Das diese Zeile decodierende UND-Glied spricht auf die Variablenfunktion $y_1 = \bar{v}_1\bar{q}_3\bar{q}_2\bar{q}_1$ an. Genau diese Bedingung ist aber erfüllt, wenn nach dem Einschalten der Versorgungsspannung die Anfangsbedingungen hergestellt sind und damit auch die Trigger G1 bis G6 den Zustand 0 ($w_1 = 0$, $v_1 = 0$) einnehmen. In dieser Stellung muß das Mikrosteuerwerk so lange bleiben, bis $v_1 = 1$ geworden ist. Solange nämlich $v_1 = 0$ bleibt, werden bei jedem Taktimpuls die D-Trigger G1, G2, G3 jeweils mit dem Zustand 0 geladen, was wiederum auf $\bar{q}_3 = \bar{q}_2 = \bar{q}_1 = 1$ führt. Damit ist dieser Zustand als stabil bewiesen. Im Zeitdiagramm (Bild 3.30) ist dieser Fall beim Taktimpuls 1 demonstriert.

Wird nun zwischenzeitlich ein positiver Impuls auf den Eingang START gegeben, so wird $w_1 = 1$, und mit der 0/1-Flanke des nächsten Taktimpulses wird auch $v_1 = 1$. Der Trigger G5 ist deswegen eingesetzt, damit die willkürlichen Zeitereignisse START oder auch Interruptgabe zu einem takt-

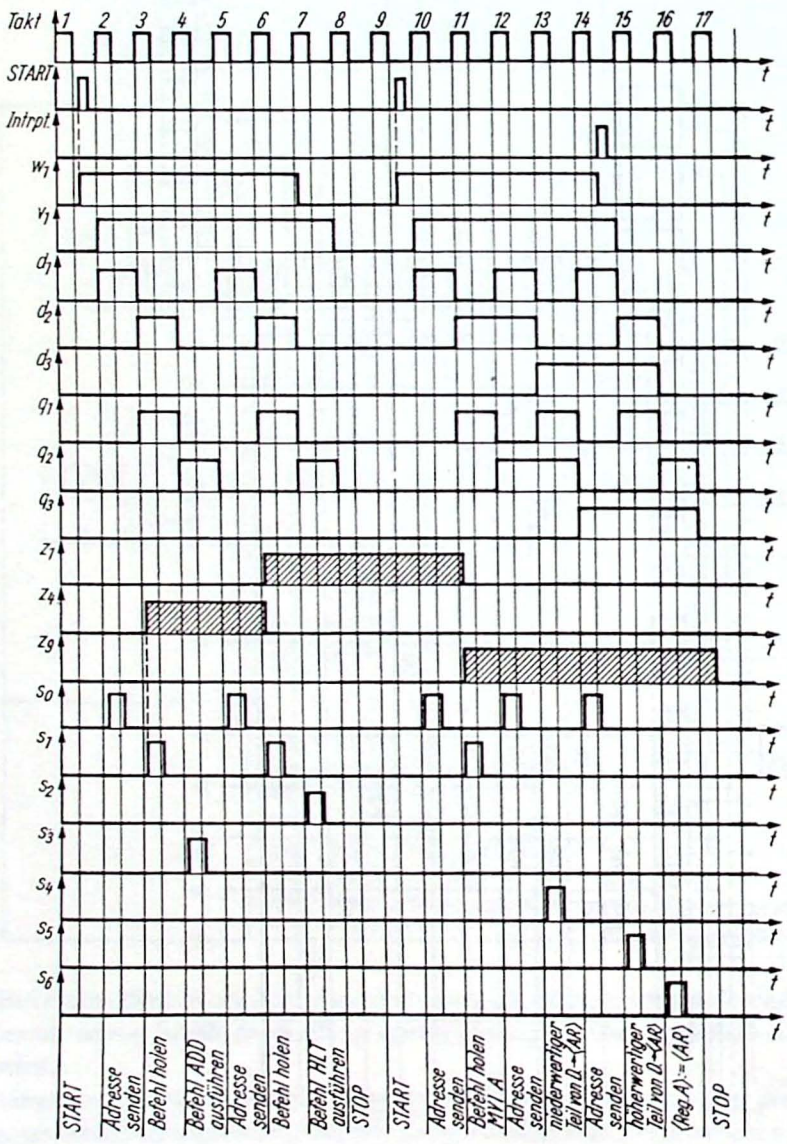


Bild 3.30. Zeitdiagramme zur Erläuterung der Wirkungsweise des Mikrosteuerwerks nach Bild 3.29

Tafel 3.13. Variablendefinition zum Mikrosteuerwerk nach Bild 3.29

Zustands-variable	Zustand
$v_1 = 0$	Der Mikroprozessor wurde durch Herstellen des Anfangszustands, durch Interruptsignal oder durch den Befehl HLT in den STOP-Zustand gebracht.
$v_1 = 1$	Der Mikroprozessor befindet sich im gestarteten Zustand
$v_2 = 0$	Es liegt keine Warte-Anforderung vor.
$v_2 = 1$	WAIT-Signal vorhanden. Der Mikroprozessor wartet mit der weiteren Abarbeitung, bis das WAIT-Signal zurückgenommen wird

Steuersignal	Wirkung
s_0	Der Stand des Programmzählers wird als Adresse zum Befehlsspeicher (ROM) übertragen. Anschließend wird der Programmzählerstand um eins erhöht
s_1	Der Inhalt des adressierten ROM-Platzes ist ein Befehl und wird in das Befehlsregister (BR) übertragen
s_2	Nach dem „Erkennen" des Befehls HLT wird der Schaltzustandstrigger G_6 zurückgesetzt
s_3	Das Ergebnis der Addition der Inhalte von Reg. A und Reg. B wird in den Akkumulator eingespeichert
s_4	Der niederwertige Teil eines Direktoperanden oder einer Sprungadresse wird in das Auffangregister geladen
s_5	Der höherwertige Teil eines Direktoperanden oder einer Sprungadresse wird in das Auffangregister geladen
s_6	Der Inhalt des Auffangregisters wird in den Akkumulator übertragen
s_7	Der Inhalt des Auffangregisters wird als Zählerstand in den Programmzähler eingespeichert

synchronen Ereignis umgewandelt werden. Die gleiche Funktion hat der Trigger G4 bei der WAIT-Signalgabe.

Betrachten wir noch einmal die Situation kurz vor Eintreffen des zweiten Taktimpulses (Bild 3.30). Die Adresse d ist immer noch $d = 000$, so daß mit der 0/1-Flanke des Taktimpulses 2 dieser Wert wieder zur Bildung der Variablen q herangezogen wird. Die Variable q bleibt also $q = 000$; aber v_1 ist jetzt im Zustand 1. Damit wird Gatter G8 aktiv und versetzt y_2 in den Zustand 1. Nach Programmablaufplan des Mikrosteuerwerks muß jetzt der Programmzählerstand zum Adreßregister des ROM übertragen und anschließend um eins erhöht werden. Das wird über den Steuerimpuls s_0 ausgelöst. Dabei muß dafür gesorgt werden, daß zunächst der Stand des Programmzählers auf den Bus gegeben wird, daß dann das externe Adreßregister für den ROM den Programmzählerstand abspeichert und danach der Impuls zur Erhöhung des Programmzählerstands gegeben wird.

Diese drei Maßnahmen müssen in entsprechenden Zeitabständen nacheinander realisiert werden. Das erfordert die Bereitstellung weiterer Impulsflanken innerhalb einer Taktperiode (Mehrphasentakt). Diese einzelnen zeitlichen Abläufe werden hier aus Vereinfachungsgründen nicht betrachtet. Die Ausgangsimpulse s_0 bis s_7 des Mikrosteuerwerks stellen also nur die Enable-Signale für das Wirksamwerden der speziellen Impulse des Mehrphasentakts an den jeweils entsprechenden Stelleingängen dar.

Mit $y_2 = 1$ wird $d_1 = 1$, und damit lautet nach der 0/1-Flanke des nächsten Taktimpulses (Taktimpuls 3) die Variable $q = q_3q_2q_1 = 001$. Ist nun eine WAIT-Anforderung gegeben, d.h. $v_2 = 1$, so wird G10 durchgeschaltet. Das hat wieder eine Selbsthaltung des Zustands zur Folge, da wiederum $d_1 = 1$ und damit $q = 001$ gebildet wird (im Bild 3.30 nicht dargestellt). Erst wenn die WAIT-Anforderung zurückgezogen worden ist ($v_2 = 0$), wird bei gleichem q-Wert Gatter G9 aktiv

und führt auf $y_3 = 1$. Damit wird $s_1 = 1$, so daß der Inhalt des adressierten ROM-Platzes (hier ein Befehl) geholt und in das Befehlsregister (BR) eingespeichert und vom Befehlsdecoder (stilisiert dargestellt) entschlüsselt wird. Gleichzeitig wird $d = 010$ gebildet.
Das führt mit der 0/1-Flanke des nächsten Taktimpulses zur Aktivierung desjenigen Gatters, das die Ausführung des aufgerufenen Befehls steuern soll. In unserem Beispiel soll das der Befehl ADD B sein ($z_4 = 1$). Damit wird G12 aktiv und $y_6 = 1$. Durch den Steuerimpuls s_3 wird die ALU auf Addition programmiert und das gebildete Ergebnis in den Akkumulator Reg. A übertragen.
Die mit $y_6 = 1$ gebildete Adreßinformation lautet $d = 000$, so daß mit der 0/1-Flanke des nächsten Taktimpulses (Impuls 5) wiederum Gatter G8 aktiviert wird und die nächste ROM-Adresse vom Programmzähler gesendet wird. Kommt nun ein anderer Ein-Wort-Befehl an die Reihe, wiederholt sich alles in der gleichen Folge. Nur die Befehlsausführung erfolgt mit einem anderen Gatter und einer anderen Steuervariablen.
Im Beispiel (Bild 3.30) folgt der Haltbefehl. Dann wird beim Schritt „Befehl HLT ausführen" (Impuls 7) Gatter G11 aktiv. Damit wird $s_2 = 1$, und der Trigger G6 wird umgeschaltet ($w_1 = 0$). Als Adreßinformation liegt mit $y_5 = 1$ $d = 000$ vor. Mit der 0/1-Flanke des nächsten Taktimpulses wird durch Trigger G5 $\bar{v}_1 = 1$, und durch die Trigger G1 bis G3 wird $q = 000$. Das führt zum Einlauf in die STOP-Schleife mit $y_1 = 1$. Das ist aber die bereits beschriebene Anfangslage.
Vor dem 10. Taktimpuls soll wieder eingeschaltet werden, so daß mit dem 10. Taktimpuls $y_2 = 1$ und damit wieder der neue Programmzählerstand gesendet wird. Mit Taktimpuls 11 soll der Befehl MVI A geholt werden, wodurch $z_9 = 1$ wird. Nach *Erkennen* des Befehls MVI A müssen die Inhalte der nächsten beiden ROM-Plätze in das Auffangregister (AR) transportiert werden. Also muß im Takt 12 der nächste Programmzählerstand zum Adreßregister des ROM gesendet werden. Im Takt 13 (Gatter 14 aktiv) wird der entsprechende ROM-Platz gelesen und durch Steuerimpuls s_4 als niederwertiger Teil in das Auffangregister übertragen. Im Takt 14 wird wieder der inzwischen um eins erhöhte Programmzählerstand zum Adreßregister des ROM gesendet (Gatter G15 aktiv), und mit Taktimpuls 15 (Gatter G16 aktiv) wird der höherwertige Teil des Direktoperanden in das Auffangregister eingeschrieben. Damit kann dann im Takt 16 (Gatter G17 aktiv) der Inhalt des Auffangregisters in das Zielregister, im Fall des Befehls MVI A in den Akkumulator Reg. A, übertragen werden.
Vielleicht haben Sie im Bild 3.30 bemerkt, daß zwischen Taktimpuls 14 und 15 ein Interruptimpuls gegeben worden ist. Dieser führt sofort zu $w_1 = 0$ und mit der 0/1-Flanke des Taktimpulses 15 auch zu $v_1 = 0$. Diese Umschaltung wirkt sich aber erst dann aus, wenn im Takt 17 mit $q = 000$ und $v_1 = 0$ die STOP-Schleife ($y_1 = 1$) aktiviert wird.
Auf diese Weise wird erreicht (das gilt auch für alle anderen Befehlsabläufe), daß nach Interruptgabe der gerade laufende Befehl noch vollständig abgearbeitet wird, ehe der Mikroprozessor mit der Programmabarbeitung aufhört. Auf diese Weise werden undefinierte STOP-Zustände vermieden.
Im Bild 3.29 ist noch der Signalflußplan für die Abarbeitung des Befehls JNZ dargestellt. Auf eine Darstellung im Zeitdiagramm wollen wir verzichten.
Der Befehl JNZ soll nur dann ausgeführt werden, wenn der Zero-Trigger nicht gesetzt ist. Ist das der Fall, so müssen nach *Erkennen* des Befehls JNZ durch das Mikrosteuerwerk die Inhalte der beiden nächsten ROM-Plätze in das Auffangregister gebracht und von dort zum Zielregister, in diesem Fall der Programmzähler, weitergeleitet werden. Damit erhält der Programmzähler den beabsichtigten Zählerstand, und die weitere Befehlsabarbeitung erfolgt von dieser Ansprungstelle aus.
Ist jedoch der Zero-Trigger gesetzt, darf der Sprung nicht ausgeführt werden. Wir wollen das in der Weise realisieren, daß wir zwar das Mikrosteuerwerk so laufen lassen, als ob der Sprung ausgeführt werden sollte, aber den Transport der Sprungadresse vom Auffangregister zum Programmzähler nicht zustande kommen lassen. Wir führen diesen Stellbefehl über ein Tor, das in Abhängigkeit vom Stand des Zero-Triggers geöffnet wird. Ist der Zero-Trigger nicht gesetzt, ist das Tor geöffnet; ist das Zero-Flag gesetzt, ist das Tor geschlossen.

Da der Befehl JNZ bis auf den letzten Schritt genauso abläuft wie der Befehl MVI A, werden bei seiner Abarbeitung ebenfalls die Gatter G13, G14, G15, G16 in der aufgeführten Reihenfolge aktiviert, ehe dann zuletzt mit der Durchschaltung von G18 der Transport der Adresse vom Auffangregister zum Programmzähler vorgenommen wird.

Zum Abschluß dieser Betrachtungen sollten Sie überlegen, welche Funktion das ODER-Glied G19 hat.

Antwort: Es sorgt mit dafür, daß bei allen Drei-Wort-Befehlen die Gatter G13 bis G16 nacheinander aktiviert werden, indem es beim Adressenstand $q = 010$ den vorgenannten Ablauf einleitet.

Zustandssignale. In der Aufgabenstellung zu unserem einfachen Mikroprozessor haben wir auf S. 159 noch gefordert, daß das Mikrosteuerwerk die jeweils eingenommenen Zustände nach außen melden soll.

Die Erzeugung dieser Signale (im Bild 3.29 nicht dargestellt) ist kein Problem, da sie durch einige Spaltenleitungen und die entsprechenden Knotenpunktdioden erfolgen kann. Im Prinzip könnte man alle Zustandssignale, die die externen Baugruppen zur Zusammenarbeit mit dem Mikroprozessor benötigen, über je eine Leitung nach außen geben. Dann lägen sie alle in decodierter Form vor, und man brauchte keine externe Logik. Dieser Zustand läßt sich aber kaum realisieren, da die Anzahl der Anschlußstifte eines Mikroprozessors auf ein Minimum reduziert werden muß, weil nicht soviel Platz auf dem Chip für Anschlußinseln (Bondinseln) ist. Aus diesem Grunde müssen die Steuersignale meist binär codiert ausgegeben und extern dann wieder decodiert werden.

Es müssen folgende Zustände nach außen signalisiert werden:

- Adresse wird gesendet.
- Befehl ist zu lesen.
- Daten sollen ein- bzw. ausgegeben werden.
- Abarbeitung des zweiten oder dritten Wortes eines Drei-Wort-Befehls.
- Mikroprozessor befindet sich im Wartezustand.
- Mikroprozessor befindet sich im STOP.

Hierzu wollen wir keine Bitmuster entwerfen. Diese Signale können noch unterteilt werden in *Statussignale* (geben jeweils den Zustand zwischen zwei Taktimpulsen an) und in *Zyklussignale* (geben die Gruppe der zu durchlaufenden Zustände an).

Abschließend soll noch darauf hingewiesen werden, daß professionelle Steuerwerke unter Anwendung von Minimierungsverfahren bei konsequenter Ausnutzung der Gemeinsamkeiten von gleichartigen Befehlen entworfen werden. Unser Mikrosteuerwerk-Beispiel wurde dagegen nach dem Gesichtspunkt leichter Faßlichkeit konzipiert.

3.8.4. Die ersten kleinen Programme

Um die Arbeitweise unseres Mikroprozessors noch besser verstehen zu lernen, wollen wir einige kleine Befehlsfolgen betrachten. Diese sollen zum Zweck des leichteren Verständnisses ausführlich und elementar behandelt werden.

Beginnen wir mit einem einfachen Datentransport.

Beispiel 1. In einem außerhalb des Mikroprozessors befindlichen Register ist die Dualzahl 0001 0011 gespeichert. Sie soll in das Register B des Mikroprozessors transportiert werden.

Vorüberlegungen. Von einem externen Register können mit Hilfe des Befehls IN (Dualcode 0001) Daten zum Akkumulator Reg. A gebracht werden, wenn diese Daten im richtigen Zeitabschnitt auf den Bus geschaltet sind. Dazu ist erforderlich, daß das externe Register über eine Eingabetorschaltung an den Bus gekoppelt wird. Die Öffnungszeit des Eingabetors wird durch die nach außen übertragenen Zustandssignale gesteuert.

Vom Register A können die eingegebenen Daten mit dem Befehl MOV B,A (Dualcode 1010) zum

Register B transportiert werden. Um diesen Datentransport zu realisieren, müssen folgende Befehle abgearbeitet werden:

IN 0001

MOV B,A 1010.

In **Tafel 3.14** betrachten wir nun die einzelnen Abarbeitungsschritte und ihre Auswirkungen.

Tafel 3.14. Erläuterung der Abarbeitung des Programmbeispiels 1

Schritt Nr.	Befehlscode		Ext. Reg.	Reg. A	Reg. B
	Mnemonik	Dualcode			
0	–	–	00010011	–	–
1	IN	0 0 0 1	00010011	00010011	–
2	MOV B,A	1 0 1 0	00010011	00010011	00010011

Als Schritt 0 wollen wir die Ausgangssituation vor Beginn der Befehlsabarbeitung bezeichnen. Wenn ein Register einen beliebigen Inhalt haben kann, soll ein Strich als Inhaltsangabe verwendet werden. Die Befehlsfolge löst also einen Transport der Zahl 0001 0011 vom externen Register in das Register A und von dort zum Register B aus. Zum Abschluß der Befehlsfolge steht in allen drei beteiligten Registern die gleiche Zahl. Es ist also immer nur eine Kopie der Ursprungszahl gesendet worden.

Beispiel 2. Es soll die Dualzahl 0011 0000 von einem externen Register 1 in das Register B übertragen und anschließend zu der als Direktoperand aus dem ROM gelieferten Dualzahl 0000 1001 addiert werden. Das Ergebnis wird in ein externes Register 2 eingespeichert.

Vorüberlegungen. Der erste Teil der Aufgabe, Transport einer Dualzahl von einem externen Register in das Register B, ist bereits im Beispiel 1 behandelt worden. Im zweiten Teil geht es darum, eine im ROM mit dem Befehl MVI A abgelegte Dualzahl in das Register A zu übertragen. Anschließend werden die Inhalte der Register A und B addiert und das Resultat mit dem Befehl OUT an ein externes Register 2 übergeben.

Tafel 3.15. Erläuterung der Abarbeitung des Programmbeispiels 2

Schritt Nr.	Befehlsfolge		Schrittwirkung	Ext. Reg. 1	Ext. Reg. 2	Auf Reg.	Reg. A	Reg. B
	Mnemonik	Dual-code						
0	–	–	–	00110000	–	–	–	–
1	IN	0001	⟨Ext. Reg. 1⟩ →⟨Reg. A⟩	00110000	–	–	00110000	–
2	MOV B,A	1010	⟨Reg. B⟩ :=⟨Reg. A⟩	00110000	–	–	00110000	00110000
3	MVI A	1000	⟨Reg. A⟩	00110000	–	–	00110000	00110000
4		1001	:=00001001	00110000	–	– 1001	00110000	00110000
5		0000		00110000	–	00001001	00001001	00110000
6	ADD B	0011	⟨Reg. A⟩ := ⟨Reg. A⟩ + ⟨Reg. B⟩	00110000	–	00001001	00111001	00110000
7	OUT	0010	⟨Ext. Reg. 2⟩ :=⟨Reg. A⟩	00110000	00111001	00001001	00111001	00110000

Sehen wir uns nun **Tafel 3.15** an, in der die nacheinanderfolgenden Befehle und ihre Wirkungen dargestellt sind. Wie bereits im Beispiel 1 gezeigt, ist nach Ablauf der Befehlsfolge IN, MOV B,A die gleiche Dualzahl in den drei Registern A, B und ext. Reg. 1 vorhanden. In diesem Fall 0011 0000. Das ist der Zustand nach Durchführung von Schritt 2. Im Schritt 3 wird der Befehl MVI A (Dualcode 1000) gegeben. Dieser hat zur Folge, daß die Inhalte der nächsten beiden ROM-Plätze nicht als Befehl, sondern als niederwertiger und höherwertiger Teil eines 8 Bit breiten Direktoperanden gewertet werden. Diese beiden Teile werden zunächst in dem Auffangregister gesammelt und dann als gesamtes Datenwort zum Register A übertragen.
Mit Schritt 6 erfolgen die Addition der beiden in den Registern A und B stehenden Zahlen und die Ablage der Summe im Register A. Mit Schritt 7 wird das im Akkumulator stehende Ergebnis an das externe Register 2 übergeben.
Im **Bild 3.31** ist das Zusammenspiel dieses Programms (Hauptprogramm) mit dem Programm des Mikrosteuerwerks dargestellt.
Bei der *Abarbeitung* dieses Bildes beginnen Sie beim Operationszeichen START des Programmablaufplans für das Mikrosteuerwerk. Sie gelangen dann zum Schritt *Adresse senden.* Als erste

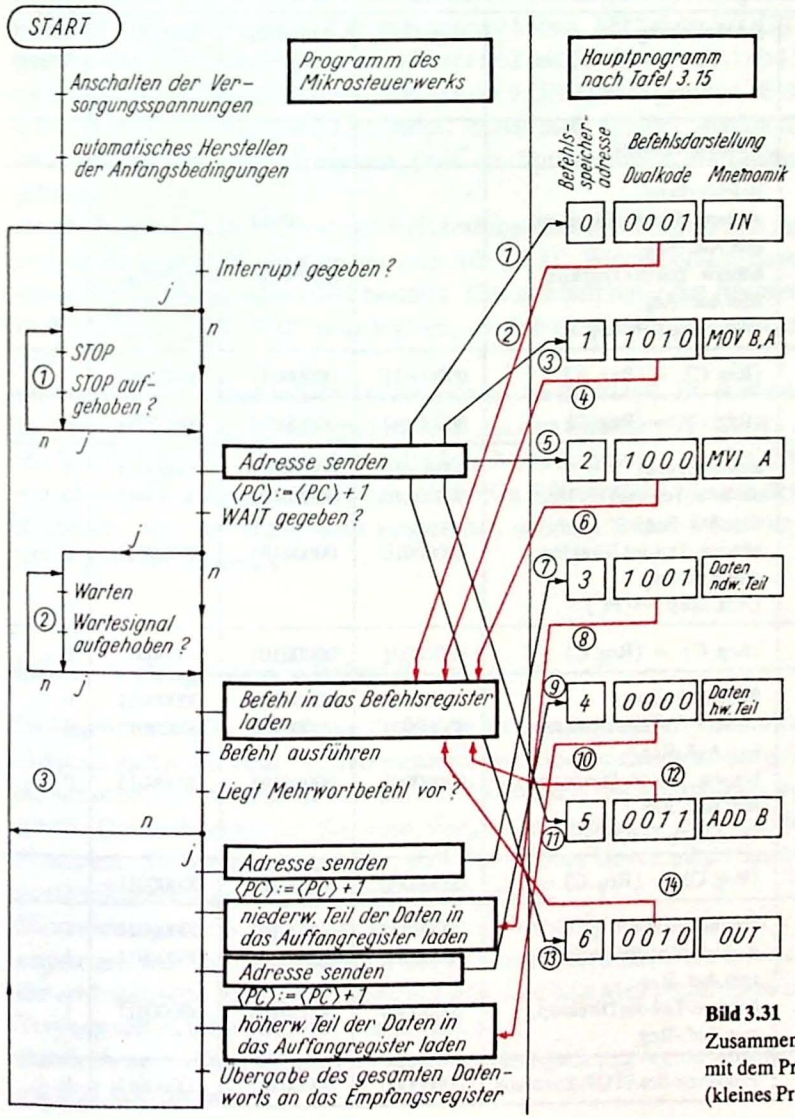


Bild 3.31
Zusammenspiel des Hauptprogramms mit dem Programm des Mikrosteuerwerks (kleines Programmbeispiel 2)

Tafel 3.16. Erläuterung der Abarbeitung des Programmbeispiels 3
a) Programmniederschrift

Befehl in Mnemonik	Befehl im Dualcode	Speicherplatz Nr.
MVI A	1000*)	1
	0011	2
	0000	3
MOV C,A	1011	4
DCR C	0101	5
JNZ	1101	6
	0101	7
	0000	8
HLT	0000	9

*) Diese Bitkombinationen müssen der Reihe nach abgespeichert werden und bilden die eigentliche Befehlsfolge.

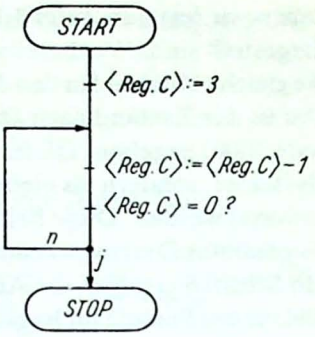


Bild 3.32
Programmablaufplan
für das kleine Programmbeispiel 3

b) ausführliche Beschreibung

Schritt Nr.	Befehlsfolge Mnemonik	Befehlsfolge Dual-code	Schrittwirkung	Reg. C	Auf.-Reg.	Reg. A	Zero-Trigger
0	–	–	Ausgangszustand	–	–	–	–
1	MVI A	1000	Befehl erkannt	–	–	–	–
2		0011	niederw. Teil des Direktop. zum Auf.-Reg.	–	– 0011	–	–
3		0000	höherw. Teil des Direktop. zum Auf.-Reg. ⟨Auf.-Reg.⟩→⟨Reg. A⟩	–	00000011	00000011	–
4	MOV C,A	1011	⟨Reg. C⟩ := ⟨Reg. A⟩	00000011	00000011	00000011	–
5	DCR C	0101	⟨Reg. C⟩ := ⟨Reg. C⟩ − 1	00000010	00000011	00000011	0
6	JNZ	1101	Befehl erkannt	00000010	00000011	00000011	0
7		0101	niederw. Teil des Direktop. zum Auf.-Reg.	00000010	00000101	00000011	0
8		0000	höherw. Teil des Direktop. zum Auf.-Reg. ⟨Auf.-Reg.⟩→⟨PC⟩	00000010	00000101	00000011	0
5/1	DCR C	0101	⟨Reg. C⟩ := ⟨Reg. C⟩ − 1	00000001	00000101	00000011	0
6/1	JNZ	1101	Befehl erkannt	00000001	00000101	00000011	0
7/1		0101	niederw. Teil des Direktop. zum Auf.-Reg.	00000001	00000101	00000011	0
8/1		0000	höherw. Teil des Direktop. zum Auf.-Reg. ⟨Auf.-Reg.⟩→⟨PC⟩	00000001	00000101	00000011	0
5/2	DCR C	0101	⟨Reg. C⟩ := ⟨Reg. C⟩ − 1	00000000	00000101	00000011	1
6/2	JNZ	1101	Befehl erkannt	00000000	00000101	00000011	1
7/2		0101	niederw. Teil des Direktop. zum Auf.-Reg.	00000000	00000101	00000011	1
8/2		0000	höherw. Teil des Direktop. zum Auf.-Reg.	00000000	00000101	00000011	1
9	HLT	0000	Einnahme des STOP-Zustands	00000000	00000101	00000011	1

Adresse erscheint Null. Das führt zum Befehl IN, der in das Befehlsregister geholt, decodiert und ausgeführt wird.
Nach erfolgtem Umlauf im Programmablaufplan des Mikrosteuerwerks kommen Sie dann wieder zum Schritt *Adresse senden*. Diesmal lautet die Adresse eins und der zugehörige Befehl MOV B,A. In dieser Weise geht die Abarbeitung weiter.
Als letztes Elementarbeispiel wollen wir ein kleines Programm mit Anwendung des bedingten Sprungbefehls JNZ (Sprung ausführen, wenn der Zero-Trigger nicht gesetzt ist) kennenlernen. Die dabei auftretende Programmschleife wird oft als Programmierelement verwendet.
Beispiel 3. Die Dualzahl 0000 0011 soll als Direktoperand über Register A in das Register C eingegeben werden. Es soll bis zur Zahl 0000 0000 abwärts gezählt werden. Danach soll der STOP erfolgen.
Vorüberlegung. Es soll, wie im **Bild 3.32** dargestellt, ein zyklischer Programmablauf realisiert werden. Das ergibt die kürzeste Form der Niederschrift, und es wird auch die geringste Anzahl von Plätzen im Befehlsspeicher beansprucht.
Betrachten wir anschließend den in **Tafel 3.16** dargestellten Programmablauf. In den Abarbeitungsschritten 1 bis 4 wird ein Direktoperandentransport zum Register C durchgeführt, den wir in ähnlicher Form im Beispiel 2 kennengelernt haben. Mit Schritt 5 (DCR C) wird vom Inhalt des Registers C die Zahl eins abgezogen. Dieser sich mit den Schritten 5/1 und 5/2 wiederholende Vorgang entspricht dem Abwärtszählen. Mit Schritt 6 (JNZ) wird getestet, ob der Zero-Trigger gesetzt ist, d.h., ob er sich im Zustand 1 befindet. Ist das nicht der Fall, werden die mit den Schritten 7 und 8 ins Auffangregister übertragenen Teile der Sprungadresse abschließend in den Programmzähler geladen.
Damit wird die Abarbeitung der Befehlsfolge erneut bei Schritt 5 fortgesetzt. Zur Unterscheidung zum ersten Schritt 5 wird dieser nun mit 5/1 (1. Wiederholung) bezeichnet. Als Sprungadresse wurde die Dualzahl 0000 0101 benutzt. Das geschah unter der Annahme, daß die Befehlsfolge ab ROM-Adresse 0000 0001 abgelegt ist, so daß die Schrittnummer der Speicherplatzadresse entspricht.
Im Schritt 5/1 wird der Inhalt des Registers C zu 0000 0001. Da er immer noch nicht Null geworden ist, wird der bedingte Sprungbefehl JNZ mit den Schritten 6/1, 7/1, 8/1 noch einmal ausgeführt.
Mit Schritt 5/2 wird anschließend der Inhalt des Registers C zu 0000 0000, womit der Zero-Trigger den Zustand 1 annimmt. Dadurch wird beim Schritt 8/2 nicht mehr der Programmzähler geladen, so daß der Sprungbefehl nicht ausgeführt, sondern Schritt 9 realisiert wird. Damit ist dann der STOP-Zustand erreicht.

3.8.5. Programmierbeispiel

Im folgenden wollen wir einige Probleme kennenlernen, die bei der Realisierung eines praktischen Beispiels auftreten können. Wir nehmen uns dabei noch einmal die gleiche Aufgabenstellung vor, die wir schon im Abschnitt 3.5. bei der Einführung des Matrixsteuerwerks als Beispiel verwendet haben. Dadurch erhalten Sie eine Vergleichsmöglichkeit zwischen den Wirkungsweisen beider Prinzipien. Sie können erkennen, daß die größere Universalität des Mikroprozessors durch eine umständlichere Abarbeitung erkauft worden ist.
Formulierung der Aufgabenstellung. Von einem digitalen Meßgerät sollen fortlaufend Meßwerte empfangen werden. Zur Glättung des zu erwartenden Meßwertverlaufs soll von je vier Meßwerten der arithmetische Mittelwert gebildet und an einen Meßwertdrucker ausgegeben werden. Dieser Vorgang soll sich fortlaufend wiederholen.
Entwicklung mathematischer Modellvorstellungen. Zur Veranschaulichung der Aufgabenstellung soll **Bild 3.33** dienen.

Zu den Zeitpunkten t_i soll, von t_0 aus beginnend, der jeweilige Wert x_i der betrachteten Meßgröße gebildet und in dualer Codierung zum Mikroprozessor übertragen werden. Aus jeweils vier Meßwerten soll das arithmetische Mittel $\bar{x}_k$ gebildet und als Repräsentant dieser Werte über dem Zeitmittelwert $\bar{t}_k$ aufgetragen werden.

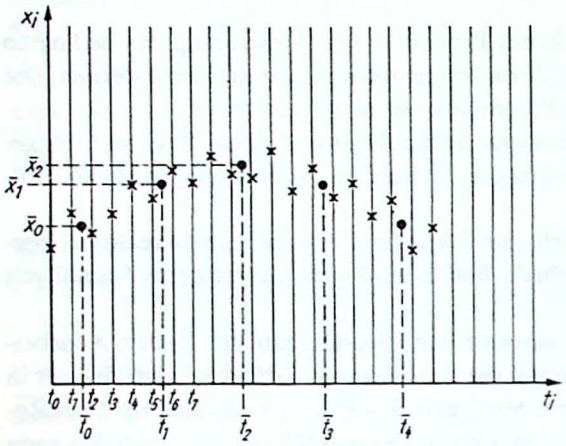


Bild 3.33
Veranschaulichung der Meßwertverarbeitung bei der Bildung des arithmetischen Mittelwerts von je vier aufeinanderfolgenden Meßwerten

Es werden folgende Beziehungen angewendet:

$$\bar{x}_k = \frac{1}{4} \sum_{i=4k}^{4k+3} x_i = \frac{x_{4k} + x_{4k+1} + x_{4k+2} + x_{4k+3}}{4}$$

$$\bar{t}_k = \frac{1}{4} \sum_{i=4k}^{4k+3} t_i = \frac{t_{4k} + t_{4k+1} + t_{4k+2} + t_{4k+3}}{4} .$$

Die Zählung der Indizes wird in **Tafel 3.17** erläutert. So geht z. B. die Zählung für $k = 3$ von $i_{\min} = 4 \times 3 = 12$ bis $i_{\max} = 4 \times 3 + 3 = 15$.

Tafel 3.17
Zuordnung der Indizes der Meßwerte zu den Indizes der Mittelwerte

k	i	
	von	bis
0	0	3
1	4	7
2	8	11
3	12	15
4	16	19
5	20	23
6	24	27
7	28	31
8	32	35
9	36	39
10	40	43

Die gebildeten Mittelwertskoordinaten können wieder in das Diagramm nach Bild 3.33 übernommen werden. Legt man mit dem Kurvenlineal eine Näherungskurve zwischen die Punkte, erhält man den geglätteten Meßwertverlauf.

Zur Programmierung nehmen wir wieder die Befehle unseres einfachen Mikroprozessors.

Als ersten Programmierungsschritt wollen wir die benötigten Speicherplätze festlegen.

Wir benötigen ein Register zum Auffangen der Meßwerte x_i. Dazu wollen wir Reg. A verwenden:

$$\langle \text{Reg. A} \rangle = a := x_i .$$

Die Summe der jeweils aufeinanderfolgenden vier Meßwerte soll im Reg. B aufbewahrt werden:

$$\langle \text{Reg. B} \rangle = b := \sum_{i=4k}^{4k+3} x_i = x_{4k} + x_{4k+1} + x_{4k+2} + x_{4k+3}.$$

Um genau vier Meßwerte aufzusummieren, benötigen wir einen Zähler. Diesen gewinnen wir, indem wir in das Register C die Zahl vier bringen.:

$$\langle \text{Reg. C} \rangle = c := 4.$$

Der Inhalt des Registers C soll dann nach der Verarbeitung jedes Einzelmeßwerts um eins erniedrigt werden. Sind vier aufeinanderfolgende Meßwerte eingegeben und aufaddiert worden, kann nach erfolgter Division durch vier das Ergebnis als arithmetischer Mittelwert ausgegeben werden. Anschließend wird der Inhalt des Registers C wieder auf die Zahl vier gebracht, und der Vorgang kann erneut beginnen.

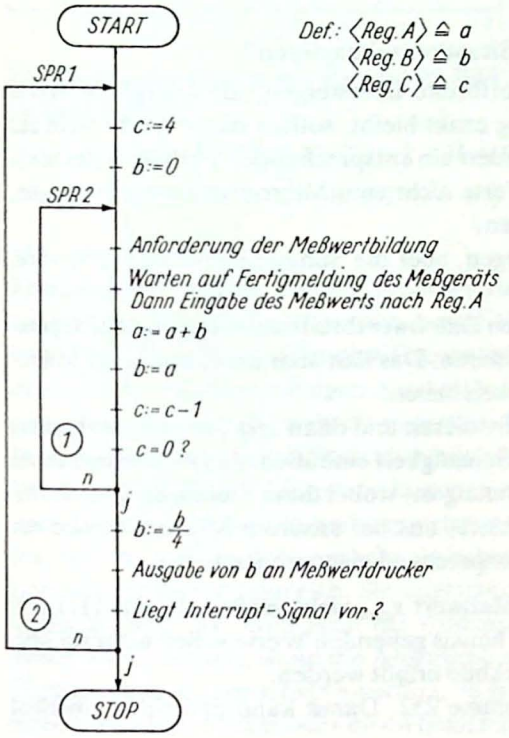


Bild 3.34
Programmablaufplan
für die Meßwerterfassung und Mittelwertbildung
von je vier aufeinanderfolgenden Meßwerten

Aus diesen Vorüberlegungen heraus wollen wir nun den Programmablaufplan **(Bild 3.34)** betrachten.

Da in der Aufgabenstellung die ständige Wiederholung des Vorgangs gefordert wird, muß zur effektiven Lösung mit Programmschleifen gearbeitet werden. In der Programmschleife 1 werden je vier aufeinanderfolgende Meßwerte eingegeben und aufsummiert. In der Programmschleife 2, in der die Programmschleife 1 enthalten ist, werden zu Beginn Register C mit der Zahl vier und Register B mit der Zahl null geladen. Nach Abarbeitung der Schleife 1 werden in der Schleife 2 noch die Division durch vier und die Ausgabe des Ergebnisses realisiert.

Ein STOP der Programmabarbeitung kann nur durch Interrupt erfolgen. Eine andere Möglichkeit wäre noch, die Fertigmeldung des Meßgeräts zu unterdrücken.

Sehen Sie sich bitte den Programmablaufplan an, und überprüfen Sie die Notwendigkeit der einzelnen Arbeitsschritte!

Beachten Sie auch die kritischen Stellen, und überprüfen Sie die Realisierungsmöglichkeit aller Schritte mit den Befehlen unseres einfachen Mikroprozessors!

Zwei Probleme müssen auf alle Fälle näher betrachtet werden:

1. Was passiert, wenn irgendein Meßwert, ein Zwischenergebnis oder das Ergebnis die vorgegebene Datenwortbreite überschreitet?
2. Wie kann die Meßwertbildung angefordert werden, und wie soll die Fertigmeldung erfolgen?

Zum ersten Problem. Diese Frage führt uns auf eine wichtige Problematik, die der Programmierer beim heutigen Stand der Technik immer beachten muß.
Der Programmierer muß bei allen Rechenoperationen den zulässigen Zahlenbereich festlegen.
Wir wollen der Einfachheit halber verabreden, daß wir nur ganze Zahlen von der Breite unseres 8-Bit-Datenworts zulassen. Das bedeutet, daß alle ganzen Zahlen von 0 bis 255 auftreten dürfen.
Wird nun die Breite des 8-Bit-Datenworts durch irgendeinen Wert überschritten, so muß das Ergebnis fehlerhaft werden, da eine unbekannte Anzahl von Bitstellen nicht an der Rechnung beteiligt ist.
Welche Möglichkeiten gibt es nun, auf eine derartige Situation zu reagieren?

a) Man könnte alle Eingabewerte, die die Datenwortbreite übersteigen, als übergroße Werte signalisieren und registrieren. Damit die Rechnung exakt bleibt, sollten diese Werte nicht zur Mittelwertbildung herangezogen werden. Das erfordert ein entsprechendes Verhalten des Meßwerterfassungssystems. Es darf die übergroßen Werte nicht zum Mikroprozessor übertragen, muß aber ein Signal an das Registriersystem abgeben.
b) Wenn alle Eingabewerte im zulässigen Bereich liegen, aber die Summe unzulässig groß wird, gibt es folgende Reaktionsmöglichkeiten:
   Man könnte mikroprozessorintern mit der doppelten Datenwortbreite arbeiten und alle Operationen (wegen der Busbreite) in zwei Schüben realisieren. Das läßt aber unser einfacher Mikroprozessor nicht zu, da wir schon alle Register vergeben haben.
   Man könnte gleich jeden Eingabewert durch vier dividieren und dann erst die Summenbildung vornehmen. Das wiederum führt dazu, daß wir an Genauigkeit einbüßen, da zur Division durch vier eine Rechtsverschiebung um zwei Stellen notwendig ist, wobei diese Stellen dann verlorengehen. Wollen wir genaue Ergebnisse haben, so bleibt uns bei unserem Mikroprozessor nur übrig, den Gültigkeitsbereich unserer Meßwerte entsprechend einzugrenzen.

Wir verabreden deshalb, daß der maximal zulässige Meßwert x_{max} nicht größer als 00111111 sein darf. Das entspricht der Dezimalzahl 63. Alle darüber hinaus gehenden Werte sollen nicht zur Mittelwertbildung zugelassen, aber ihr Auftreten im Druckbild erfaßt werden.
Wird viermal die Zahl 63 addiert, erhält man als Summe 252. Damit kann es keinen Überlauf geben.
Sehen wir uns nun das Ergebnis bei einer Mittelwertbildung von vier willkürlich angenommenen Meßwerten an:

$$\begin{array}{rr} x_1 & 58 \\ x_2 & 60 \\ x_3 & 63 \\ x_4 & 57 \\ \hline & 238 : 4 = 59{,}5. \end{array}$$

Wir stellen fest, daß neben der ganzen Zahl noch ein Rest hinter dem Komma steht.
In Dualzahlform lautet die Summe 11101110. Führt man die Division durch vier aus, erhält man:

0011 1011, 10.

Die acht Stellen vor dem Komma ergeben die Zahl 59. Die zwei Stellen nach dem Komma haben die Bedeutung von 0,5.

Da wir nur 8 Bit auswerten können, müßten die Stellen nach dem Komma verlorengehen. Doch man kann die Information dieser zwei Stellen nach dem Komma retten, indem man nicht die Zahl um zwei Stellen nach rechts, sondern das Komma um zwei Stellen nach links verschiebt:

11 1011, 10.

Ergibt das aber nicht einen Widerspruch zum verabredeten Zahlenformat? Zunächst ja, aber diese eben gebildete Zahl stellt das Ergebnis dar und wird nur dem Druckersystem mitgeteilt und nicht mehr im Mikroprozessor verarbeitet.
Man braucht also nur mit dem Druckersystem das neue Zahlenformat abzustimmen und hat auf diese Weise die größtmögliche Genauigkeit bei der Ergebnisdarstellung erzielt.

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
1	1	1	0	1	1	1	0
2^5	2^4	2^3	2^2	2^1	2^0 ,	2^{-1}	2^{-2}

Bild 3.35
Für das Druckersystem definiertes Zahlenformat

Für das Druckersystem gilt also das im **Bild 3.35** dargestellte Zahlenformat.
Im Beispiel wird

$$\bar{x}_k = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2}$$

$$\bar{x}_k = 32 + 16 + 8 + 2 + 1 + \tfrac{1}{2}$$

$$\bar{x}_k = 59.5.$$

Zum zweiten Problem. Die Anforderung zur Meßwertbildung muß vom Mikroprozessor ausgegeben werden. Wir können dazu den Ausgabebefehl OUT verwenden.
Nun haben wir schon wieder eine Grenze unseres konzipierten Mikroprozessors erreicht; denn wir haben leider nur einen einzigen Ausgabebefehl und müssen mit diesem auch noch den vom Mikroprozessor gebildeten Mittelwert ausgeben. Damit gehen alle Ausgabedaten über das gleiche Tor hinaus und sind von ihrem Bedeutungsgehalt nicht zu unterscheiden.
Nun kann natürlich der Einwand kommen, warum der Befehlssatz nicht gleich genügend umfangreich definiert wurde. Ein umfangreicherer Befehlssatz würde uns zwar im Augenblick weiterhelfen, aber wir wollten doch ein sehr einfaches Lernmodell benutzen. Und sogar die augenblickliche Kalamität ist für uns lehrreich. Sie zeigt uns deutlich, daß ein technisches System in seinen Möglichkeiten begrenzt ist. Und auch bei vollkommeneren Mikroprozessoren gibt es irgendwann Probleme, nämlich dann, wenn die Ansprüche weiter wachsen.
Darum sollten wir zunächst versuchen, ob es nicht doch Lösungsmöglichkeiten gibt, die nicht gleich einen neuen Befehlssatz oder einen besseren Mikroprozessor verlangen.
Wir haben bei der Betrachtung des ersten Problems die Festlegung getroffen, daß nur Meßwerte kleiner oder gleich 63 eingegeben werden dürfen. Danach tritt als maximale Summe von vier Meßwerten die Zahl 252 auf. Da wir aber mit 8 Bit die Zahlen von 0 bis 255 darstellen können, bleiben drei Zahlen unbenutzt. Eine davon (vorschlagsweise 255 = 1111 1111 B) können wir verwenden, damit der Mikroprozessor damit das Kommando zur Meßwertbildung geben kann.
Wir werden dann alle Ausgabewerte einem Decoder zuführen. Tritt dabei der Wert 1111 1111 auf, so wird der am Decoderausgang auftretende 0/1-Übergang zu folgenden Maßnahmen benutzt:

- Verhinderung des Drucks dieses Wertes
- Setzen eines RS-Triggers, der den WAIT-Zustand am Mikroprozessor einstellt
- Start der Meßwertbildung.

Mit der Bildung des WAIT-Zustandes im Mikroprozessor stellt dieser seine Arbeit so lange ein, bis das WAIT-Signal wieder aufgehoben ist. Ist der Meßwert gebildet und befindet er sich als digitaler Wert im Meßwertregister, so wird über einen Impuls vom Meßgerät der WAIT-Trigger rückgesetzt, und der Mikroprozessor kann seine Arbeit fortsetzen.

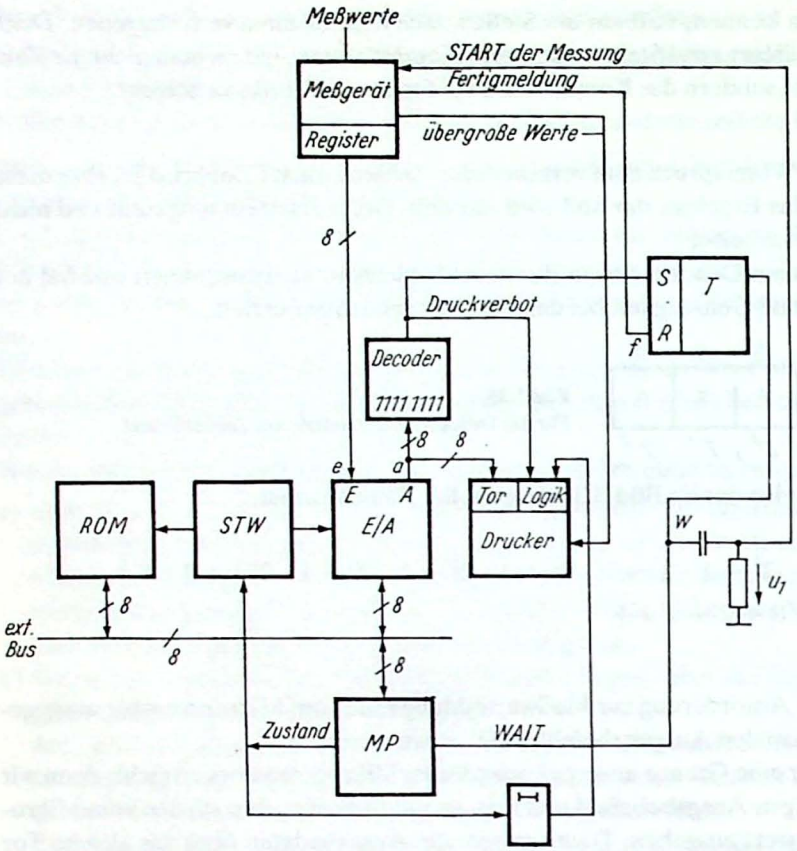


Bild 3.36. Blockbild der externen Baugruppen und Geräte zur Meßwerterfassung, Druck der Mittelwerte sowie Kennzeichnung des Auftretens übergroßer Werte im Druckbild

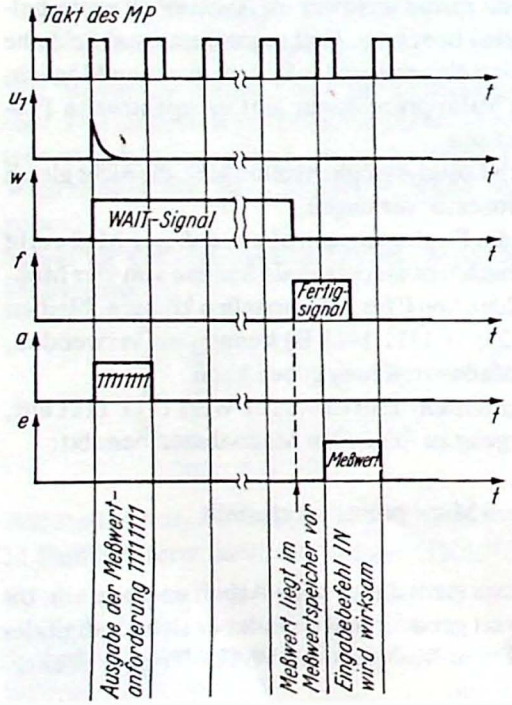


Bild 3.37
Zeitdiagramme zum Ablauf der Meßwerterfassung

Wird ein Ausgabebefehl gegeben, mit dem ein arithmetischer Mittelwert $\bar{x}_k$ ($\bar{x}_k \neq 1111\ 1111$) zum Druckersystem übertragen werden soll, reagiert der Decoder nicht, und der Mittelwert wird ausgedruckt.

Bild 3.36 zeigt eine Systemdarstellung und **Bild 3.37** das Zeitdiagramm zur Erläuterung dieses Ablaufs.

Tafel 3.18. Programm zum Programmierbeispiel „Meßwerterfassung und Mittelwertbildung"

Adresse im Befehlsspeicher	Befehl in Mnemonik	Inhalt des Speicherplatzes	Befehlswirkung
1 2 3	MVI A	1 0 0 0 0 1 0 0 0 0 0 0	⟨Reg. A⟩ := 4
4	MVI C,A	1 0 1 1	⟨Reg. C⟩ := 4
5 6 7	MVI A	1 0 0 0 0 0 0 0 0 0 0 0	⟨Reg. A⟩ := 0
8	MOV B,A	1 0 1 0	⟨Reg. B⟩ := 0 Löschen von Reg. B
9 10 11	MVI A	1 0 0 0 1 1 1 1 1 1 1 1	⟨Reg. A⟩ := 1111 1111
12	OUT	0 0 1 0	Ausgabe von 1111 1111. Damit werden das WAIT-Signal und das Kommando zur Meßwertbildung ausgelöst
13	IN	0 0 0 1	⟨Reg. A⟩ := Meßwert
14	ADD B	0 0 1 1	⟨Reg. A⟩ := ⟨Reg. A⟩ + ⟨Reg. B⟩
15	MOV B,A	1 0 1 0	⟨Reg. B⟩ := Summe der Meßwerte
16	DCR C	0 1 0 1	⟨Reg. C⟩ := ⟨Reg. C⟩ − 1 Abwärtszählen
17 18 19	JNZ	1 1 0 1 1 0 0 1 0 0 0 0	Sprung zur Adresse 9, wenn Zero-Trigger nicht gesetzt ist
20	MOV A,B	1 0 0 1	⟨Reg. A⟩ := Summe der vier Meßwerte
21	OUT	0 0 1 0	Ausgabe der Summe der vier Meßwerte zum Druck des arithmetischen Mittelwerts
22 23 24	JMP	1 1 0 0 0 0 0 1 0 0 0 0	Unbedingter Sprung zur Adresse 1. Wiederholung des gesamten Befehlsablaufs

Das aus dem Programmablaufplan auf S. 179 und allen anderen Überlegungen hervorgehende Programm ist in **Tafel 3.18** abgebildet.

Sehen Sie sich dieses Programm an, und überprüfen Sie die Wirkung aller Befehle! Vielleicht nehmen Sie dazu Papier und Bleistift zur Hand, geben sich selbst Meßwerte vor und verfolgen das Programm mit Hilfe Ihrer eigenen Rechnung.

Diese ausführlichen Betrachtungen sollten uns auf drei Umstände aufmerksam machen:

1. Das Aufstellen eines Programms erfordert viel Umsicht. Der Programmierer muß alle möglichen Abläufe und Wirkungen vorweg durchkalkulieren. Erst dann ist die Gewähr dafür gegeben, daß das Programm in allen Betriebssituationen funktioniert.

2. Es wird immer wieder Situationen geben, in denen die Grenzen einer technischen Konzeption sichtbar werden bzw. einer weiteren Verbesserung im Wege stehen. Dadurch kommt es zum Bedürfnis nach verbesserter Gestaltung des Systems.
3. Bis zu einem bestimmten Umfang kann man durch zusätzliche externe Logikschaltungen und entsprechende Programmgestaltung die Anwendungsgrenzen eines Mikroprozessors hinausschieben. Inwieweit das sinnvoll ist, muß eine entsprechende Kalkulation ergeben.

Mit diesen Darlegungen haben Sie an einem einfachen Beispiel einen Einblick in die dabei doch schon recht vielfältigen Probleme erhalten.

3.9. Programmherstellung

Wir wissen: Das Programm eines Digitalrechners stellt eine geordnete Folge von einzelnen Befehlen (Operationsanweisungen) dar. Die Befehle werden in binärer Form (als Bitmuster) in den Befehlsspeicher (ROM) eingegeben.
Dadurch wird jedem Befehl eine Speicherplatzadresse zugeordnet. Bei der Abarbeitung des Programms werden die Befehle der entsprechenden Reihenfolge nach gelesen; sie aktivieren mit Hilfe des Mikrosteuerwerks die entsprechenden Steuersignale.
Der Befehlsaufruf wird durch die Übertragung der im Programmzähler gebildeten Adresse zum Adressenauffangregister für den Befehlsspeicher (ROM) durchgeführt.

Bevor mit der Programmherstellung begonnen werden kann, bedarf es einer präzisen Formulierung der Aufgabenstellung.
Aus dieser Aufgabenstellung müssen unter Einbeziehung aller theoretischen und praktischen Kenntnisse (gegebenenfalls auch der Ergebnisse noch durchzuführender Experimente) die benötigten mathematischen Modellvorstellungen und die daraus ableitbaren Rechenvorschriften (Algorithmen) entwickelt werden.
Die erfolgreiche Lösung dieses Teilschritts setzt umfangreiche prozeßbezogene und mathematische Kenntnisse voraus. Ist der Algorithmus (der mit Methoden der numerischen Mathematik durchführbare Rechenablauf) gefunden worden, kann mit der eigentlichen Programmierung begonnen werden. Hierzu müssen folgende Unterlagen an den Programmierer übergeben werden:

- Aufgabenstellung
- Definitionsliste aller Variablen und Konstanten
- zu verwendende Formeln
- Programmablaufplan
- Zeitdiagramme
- verbale Erläuterungen zu allen Punkten.

Auch wenn die Algorithmenfindung und das Schreiben des Programms von ein und demselben Bearbeiter durchgeführt werden, sollten die aufgeführten Programmierunterlagen auf jeden Fall zur Realisierung gelangen. Dafür gibt es zwei Gründe:

- Die Herstellung dieser Unterlagen führt zu klareren Einsichten und bewahrt vor unnötigen Fehlern.
- Andere Bearbeiter können später mit relativ geringem Aufwand die Gedankengänge kontrollieren oder Anschlußprogramme leichter formulieren.

Selbstverständlich gibt es einfache Fälle, in denen man alles ohne Schwierigkeiten überblicken kann. Selbstverständlich muß man einräumen, daß die Grenzen für Überschaubarkeit je nach den

Fertigkeiten und Fähigkeiten der einzelnen Bearbeiter unterschiedlich sind. Man sollte jedoch immer daran denken, daß alle Menschen vergeßlich sind und daß sogar die Programmentwickler bei fehlenden Aufzeichnungen später in Schwierigkeiten geraten können, wenn sie sich nach längerer Zeit wieder in ihrem eigenen Programm zurechtfinden müssen.
Es gilt also hier den optimalen Weg zu finden, der bei minimalem Aufwand an zu erstellenden Unterlagen ein effektives Eindringen in die logischen Zusammenhänge eines Programms gewährleistet. Es sieht heute so aus, daß die Entwicklungskosten für die den adressierbaren Speicherumfang ausnutzenden Programme um ein Vielfaches höher sind als die Kosten für die Beschaffung des Mikrorechners. So wundert es nicht, wenn überall Anstrengungen unternommen werden, die Kosten für die Programmherstellung zu senken.
Sehen wir uns an, welche Formen der Programmniederschrift und welche Hilfsmittel zur Programmaufbereitung vorliegen.
Die Formen der Programmniederschrift sind in **Tafel 3.19** zusammengestellt.

Tafel 3.19. Formen der Programmniederschrift

Art der Niederschrift	Ausführungsbeispiel
Maschienencode in Binärzeichen	1000 0011 1110
Maschinencode in Hexadezimalzeichen	1000 → 8 0011 → 3 1110 → E } 8 3 E
Assemblersprache (Verwendung der Mnemoniks)	1000 → MVI A 0011 → 3 1110 → E } MVI A,3E H
Höhere Programmiersprache	vollständige, kleinere Programmteile werden nach definierter Sprachform programmiert

Welche Auswirkungen haben nun die einzelnen Arten? Die Niederschrift des Maschinencodes in Binärzeichen ist aufwendig und verlangt vom Programmierer eine hohe Konzentration. Ein Irrtum im *Labyrinth* der Nullen und Einsen ist sehr leicht geschehen. Die Vorteile des Verfahrens bestehen darin, daß man nur Lochstanzer oder Eingabetasten im Binärcode benötigt und daß die Anzahl der Programmspeicherplätze auf den Minimalwert gebracht werden kann.
Beim Schreiben des Maschinencodes in Hexadezimalzeichen werden jeweils *vier Bitstellen zu einem Zeichen* zusammengefaßt. Da dabei zehn Ziffernzeichen und sechs Buchstaben verwendet werden, wird neben der Verringerung der Zeichenanzahl je Befehl auch noch das Schriftbild abwechslungsreicher. Dieses Verfahren verlangt daher weniger Konzentration vom Programmierer, erfordert aber apparativen Aufwand zur automatischen Rückübersetzung in den vom Mikrorechner verlangten Binärcode. Die Anzahl der Programmspeicherplätze läßt sich ebenfalls auf den Minimalwert bringen.
Bei Anwendung der Assemblersprache kommmen die *Mnemonikausdrücke* (Mnemonics) zur Geltung. Die einzelnen Befehle müssen in einer verabredeten Form geschrieben werden. Das Schriftbild bekommt noch mehr Buchstaben: die verwendeten Mnemonics erinnern an sprachliche Formulierungen. Die Übersetzung (Assemblierung) eines in Assemblersprache notierten Programms erfordert bereits einen hohen Aufwand an Rechnerkapazität. Auch hier läßt sich die Anzahl der benötigten Programmspeicherplätze auf den Minimalwert bringen.
Beim Einsatz höherer Programmiersprachen hat der Programmierer einen größeren Aufwand beim Erlernen der Sprache, bekommt aber bei ihrer Beherrschung wesentliche Erleichterung beim

Programmieren. Dafür ist der Aufwand für das *Übersetzungsprogramm,* (Compiler) noch größer, und wegen der Zusammenfassung ganzer Programmteile kann der minimale Wert an Programmspeicherplätzen nicht mehr erreicht werden.
Die zukünftige Entwicklung wird dadurch gekennzeichnet sein, daß die Programmherstellung immer mehr dem Rechner übertragen wird. Der Rechner wird mit Hilfe immer größerer und komplizierterer Programme und durch entsprechende Wahrnehmungsglieder (Sensoren) in die Lage versetzt werden, Programmablaufpläne und Zeitdiagramme zu lesen, Formeln zu erkennen, Variable und Konstanten aus schriftlichen Textaufgabenstellungen zu entnehmen. Es ist bereits abzusehen, daß auch die akustische Eingabe mit menschlicher Sprache so gelöst werden wird, daß im Endziel der Rechner, immer selbständiger werdend, Programme aus einer kurzen, verbalen Aufgabenstellung heraus völlig in eigener Regie aufbaut. Bei entstehenden Unklarheiten wird er sogar Fragen stellen, um eindeutige Verhältnisse zu schaffen.
Noch nicht abzusehen ist, zu welchem Zeitpunkt die einzelnen Entwicklungsstadien zum Durchschnittskomfort gehören werden. Klar ist aber, daß wir am Anfang einer gewaltigen Entwicklung *intelligenter* Maschinensysteme stehen.
Doch zurück zum Alltag. Heute dominieren beim Mikrorechnereinsatz noch die Assemblersprachen, und die höheren Programmiersprachen werden immer stärker eingesetzt. Das ist bedingt durch die ständig wachsende Anzahl der adressierbaren Speicherplätze.
Anliegen dieses Buches kann es nicht sein, eine höhere Programmiersprache zu erläutern. Jedoch der Prozeß, der bei Einsatz einer Assemblersprache vollzogen wird, soll zur Einführung in die Problematik näher betrachtet werden.

3.9.1. Programmherstellung mit Hilfe der Assemblersprache

Das Wort *assembler* kommt aus dem anglo-amerikanischen Sprachbereich und bedeutet *Monteur.* Das zugehörige Verb *to assemble* hat die Bedeutung von aufmontieren, (aus Teilen) zusammensetzen.

Demzufolge könnte die Assemblersprache als eine Montagesprache bezeichnet werden. Sie ist in der Tat eine aus Einzelelementen (vorwiegend Mnemonics) zusammengesetzte künstliche Sprache. Sie ist eigens zu dem Zweck geschaffen, um mit relativ geringem Aufwand erlernbar zu sein und mit nicht allzu hohem Aufwand die Übersetzung des Programms in den Maschinencode zu ermöglichen.

Tafel 3.20. Tabelle zur Programmniederschrift in Assemblersprache

Name	Operation	Variable	Kommentar
PR0G:	MVI	A,3EH	Lade Reg. A mit der Binärzahl, die der Hexadezimalzahl 3E entspricht
	ADD	B	Addiere den Inhalt von Reg. B zum Inhalt von Reg. A

Wie jede andere Sprache macht auch die Assemblersprache eine Entwicklung mit, da sie stets an die technischen Gegebenheiten angepaßt werden muß. So sind im Laufe der Zeit neue Mnemonics für neue Operationsanweisungen entstanden, ältere Bezeichnungen sind durch andere ersetzt worden, um sprachliche Präzisierungen zu ermöglichen.
Trotz der Erweiterungen und Änderungen hat sich aber das Grundgerüst der Notierung in Tabellenform erhalten **(Tafel 3.20)**.

Die erste Spalte der Tabelle trägt die Überschrift *Name.* Hier werden Namen als Repräsentanten für Adressen eingetragen. Diese Adreßnamen werden deshalb gegeben, weil man die genaue Binäradresse eines Befehlsspeicherplatzes zu Beginn einer Programmierung noch nicht kennt. Das hat dabei folgenden Vorteil: Selbst wenn im Zustand der Niederschrift des Programms nachträglich noch andere Befehle dazwischengeschoben werden müssen, kann der Adreßname erhalten bleiben; denn die Zuweisung der konkreten Binäradressen erfolgt erst bei der Assemblierung.
Über die Art und Weise, wie die Namen geschrieben werden müssen, gibt es keine einheitlichen Vorschriften. Für die Namensspalte wird die maximale Anzahl der Zeichen festgelegt, und es gibt Beschränkungen in der Wahl des ersten und letzten Zeichens. So kann man z. B. die Forderung antreffen, daß als letztes Zeichen unbedingt der Doppelpunkt gesetzt werden muß.
In die zweite Spalte wird die mnemotechnische Abkürzung der *Operationsanweisung* geschrieben. Dabei werden die Angaben über Sende- und Empfangsregister nicht mitgeschrieben, sondern kommen mit in die dritte Spalte, in der die *Variablen* und auch die *Direktoperanden* aufgeschrieben werden.
In die Kommentarspalte kann die Erläuterung der *Befehlswirkung* geschrieben werden. Im Beispiel ist dies sehr ausführlich gemacht worden. Man kann einen solchen Kommentar wesentlich kürzen, man braucht ihn auch überhaupt nicht zu schreiben. Er dient lediglich zur Erinnerung für den Programmierer oder als Verständnishilfe für andere. Bei der Assemblierung entnimmt der Übersetzungsrechner Informationen nur aus den ersten drei Spalten. Der Inhalt der Kommentarspalte wird im Übersetzungsprotokoll ohne Veränderung ausgedruckt, also kopiert.
Zur Erleichterung der Korrektur wird das Programm zunächst mit Bleistift geschrieben. Ist dann der Programmierer der Ansicht, daß alles korrekt notiert ist, wird das gesamte Programm über die Tastatur in einen entsprechenden Automaten eingegeben. Das ist entweder ein ORG-Automat oder ein Mikrorechner-Entwicklungssystem.
Beim Entwicklungssystem erfolgt die Ausgabe der Niederschrift auf dem Bildschirm. Jeder Befehl wird sofort übersetzt, falls er richtig niedergeschrieben war. Bei fehlerhafter Schreibweise gibt es eine *Fehlerkennung*, und der Automat wartet auf eine verbesserte Eingabe.
Beim ORG-Automaten erhält man ein mit Maschine geschriebenes Programmexemplar und einen Lochstreifen, den sog. *Quellenstreifen.*
Bei dieser Lochstreifenherstellung muß man wieder einige Spielregeln beherrschen. Nach dem jeweils letzten Zeichen einer Spalte (mit Ausnahme der letzten Spalte) muß unbedingt ein Tabulatorsprung ausgelöst werden. Das hat für das Druckbild der Schreibmaschine den Vorteil, daß die ersten Buchstaben jeder Spalte genau untereinander stehen. Für das Programm zur Übersetzung in den Maschinencode hat die beim Tabulatorsprung ausgestanzte Lochkombination die Bedeutung einer *Endemarke*. Dem Automaten wird dadurch mitgeteilt, daß der Zeileninhalt der vorhergehenden Spalte vollständig ist und daß mit dem nächsten Zeichen der Zeileninhalt der nächsten Spalte beginnt.
Die letzte Spalte wird in jeder Schreibzeile mit Wagenrücklauf und Zeilensprung (newline) abgeschlossen. Für den Abschluß des gesamten Programms gibt es wiederum bestimmte Regeln, z. B. das Wort END in der Operationsspalte mit anschließendem newline.
Es hat an dieser Stelle keinen Sinn, auf alle Einzelheiten einzugehen, weil jeder, der in Assemblersprache programmieren will, sich in alle speziellen Vorschriften des von ihm verwendeten Systems einarbeiten muß. Bemerkenswert ist noch, daß alle Vorschriften exakt eingehalten werden müssen, weil z. Z. nahezu kein Spielraum für Eingabevarianten vorgegeben ist. Sicherlich wird sich das mit der Weiterentwicklung ändern. Aber jede Erweiterung zu größerem Spielraum erfordert mehr Speicherkapazität; denn dadurch lassen sich mehr Wissensfakten bei den Automaten einspeichern.
Ein Beispiel für die Niederschrift des Befehls ADD B im Quellenstreifen ist im **Bild 3.38** dargestellt.

Der Quellenstreifen muß zur Assemblierung in den Übersetzungsrechner eingegeben werden. Das kann der Mikrorechner selbst sein, wenn sein Speicherumfang hinreichend groß ist. Vielfach nimmt man jedoch dazu einen anderen Rechner als denjenigen, für den das Programm gemacht worden ist. Diesen anderen Rechner nennt man *Wirtsrechner* (host-computer).

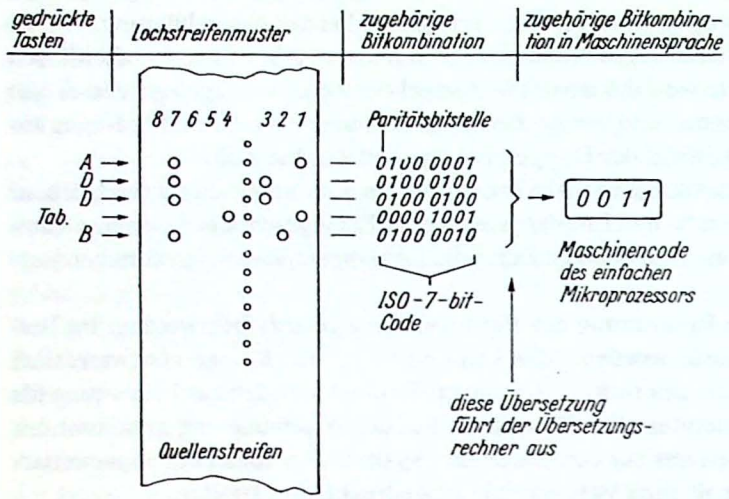


Bild 3.38
Veranschaulichung der Umsetzung des Lochmusters im Quellenstreifen in die Bitkombination des Maschinencodes

Im Ergebnis der Assemblierung (im Fall der Wirtsrechneranwendung spricht man von *Crossassemblierung*) wird ein Lochstreifen ausgegeben, der das Maschinenprogramm enthält. Außerdem entsteht ein Druckprotokoll, in dem man folgende Angaben findet:

- Niederschrift des Maschinenprogramms, z. B. in Hexadezimalcode
- die den einzelnen Befehlen vom Übersetzungsprogramm zugeordneten Speicherplatzadressen
- Abschrift des Programms in Assemblersprache (einschließlich des Kommentars)
- eine laufende Nummer für jede Schreibzeile
- Kennzeichnung und Anzahl der syntaktischen Fehler.

Unter *syntaktischen Fehlern* versteht man solche, die einen Verstoß gegen die Verabredungen der Assemblersprache (z. B. nichtauswertbare Zeichen) darstellen, oder logische Fehler einfacher Art wie mehrfache Vergabe eines Speicherplatznamens oder Angabe einer nichtdefinierten Sprungadresse.

Sind solche Fehler vorhanden, müssen sie zuerst beseitigt werden. Danach kann das Programm auf dem Wirtsrechner einem *Logiktest* unterzogen werden. Dazu wird ein Hilfsprogramm benötigt, das die Arbeitsweise des eigentlichen Mikrorechners zu simulieren gestattet. Erkennt man dabei logische Fehler (z. B. falsche Programmierung einzelner Arbeitsschritte), müssen auch diese beseitigt werden. Zur effektiven Fehlerbeseitigung können Debugger-Programme eingesetzt werden.

Die Simulation auf dem Wirtsrechner hat den Nachteil, daß der zeitliche Ablauf nicht dem des eigentlichen Rechners entspricht. Ein *Echtzeittest* kann nur mit dem zum Einsatz kommenden Mikrorechner realisiert werden. Dazu gibt man die gesamte Befehlsfolge zunächst in den RAM, weil mit diesem Speicher notwendig werdende Veränderungen in der Befehlsfolge ohne größere Umstände möglich sind. Erst wenn auch dieser Echtzeittest erfolgreich bestanden wurde, kann die Programmierung als gelungen bezeichnet werden. Den Abschluß bildet dann die Eingabe der erfolgreich getesteten Befehlsfolge in den Festwertspeicher (ROM, PROM, EPROM).

Bei Benutzung eines Mikrorechner-Entwicklungssystems können Logik- und Echtzeittest wesentlich effektiver realisiert werden. Dazu wird das mit dem zu testenden Programm versehene Anwendersystem über einen Testadapter mit dem Entwicklungssystem verbunden. Nun kann das zu testende System unter Kontrolle des Entwicklungssystems laufen. An vorgegebenen Testpunkten

kann das Entwicklungssystem alle wichtigen Daten vom Anwendersystem abfragen und auf dem Bildschirm ausgeben.
Treten nicht erklärbare Echtzeitprobleme auf, so kann mit Hilfe eines *Logikanalysators* die Klärung erleichtert werden. Diese komfortable und zeitoptimale Testsystemgestaltung hat natürlich einen entsprechenden Anschaffungspreis.
Es gibt Logikanalysatoren mit folgenden zeitlichen Auflösungsbereichen und entsprechenden Darstellungen auf dem Bildschirm:

- höchste zeitliche Auflösung: Spikes im Bereich einiger Nanosekunden
- mittlere zeitliche Auflösung: Impulsverläufe der einzelnen Signale
- grobe zeitliche Auflösung: Rückübersetzungsprotokolle aus dem Maschinencode in Assembler- oder Hochsprache.

Abschließend wollen wir die Darstellung des Programms für die Mittelwertbildung von je vier Meßwerten in Assemblersprache betrachten **(Tafel 3.21)**, das wir vorher schon in Tafel 3.18 in allgemeinerer Form zusammengestellt haben.
Sie werden sicherlich feststellen, daß die Notierung des Programms jetzt wesentlich übersichtlicher als vorher ist.

Tafel 3.21. Programm in Assemblersprache zum Programmierbeispiel „Meßwerterfassung und Mittelwertbildung"

SPR 1:	MVI	A,4	
	MOV	C,A	c := 4
	MVI	A,0	
	MOV	B,A	b := 0
SPR 2:	MVI	A,11111111B	
	OUT		Aufforderung zur Messung
	IN		Eingabe des Meßwerts
	ADD	B	$a := a + b$
	MOV	B,A	$b := a$
	DCR	C	
	JNZ	SPR 2	
	MOV	A,B	
	OUT		Ausgabe des Mittelwerts
	JMP	SPR 1	

Die Direktoperanden können sowohl in Dezimalform als auch in Binär-, Hexadezimal- oder Oktalform geschrieben werden. Mit Ausnahme der Dezimalform muß bei allen anderen ein *Kennungszeichen* mitgeschrieben werden (z. B. 1111 1111 B).
Bei den Sprungbefehlen sind die Sprungadressen jetzt mit symbolischen Namen geschrieben (SPR 1 und SPR 2). An der Stelle, an der nach Ausführung des Sprunges die weitere Abarbeitung des Programms fortgesetzt werden soll, muß dann in die Namensspalte ebenfalls diese symbolische Adresse geschrieben werden. Beim Assemblieren werden dann die symbolischen Adressen durch binäre Adressen ersetzt.

3.9.2. Unterprogrammtechnik

Wenn im Rahmen größerer Programme einzelne Programmteile mehrfach angewendet werden, kann man diese wiederkehrenden Programmteile zu *Unterprogrammen* erklären.
Das ergibt den Vorteil, daß man diese Befehlsfolgen jeweils nur einmal abzuspeichern braucht, folglich Speicherplätze einspart (vgl. S. 138).
Im **Bild 3.39** ist das Gerüst eines Programmablaufplans mit zwei Unterprogrammen (U_1 und U_2) dargestellt. Zu Unterprogrammen können z. B. die Algorithmen zur Berechnung bestimmter Funktionen, wie $y = \sqrt{x}$, $y = \sin x$ usw., erklärt werden.

Da nun ein Unterprogramm irgendwo im Hauptprogramm benötigt wird, muß es für diese ortsbeliebige Verwendung vorbereitet sein. Am günstigsten ist es, wenn man die zu verarbeitenden unabhängigen Variablen in verabredete Register des Notizbuchspeichers eingibt und sie von dort durch das Unterprogramm zur Verarbeitung abgeholt und die Ergebnisse des Unterprogramms in verabredete Übergaberegister abgelegt werden.
Befinden sich die Eingangsvariablen in den Übergaberegistern, so wird das benötigte Unterprogramm mit dem CALL-Befehl angefordert. Dieser Befehl enthält neben der Anweisung zum Unterprogrammaufruf auch die Anfangsadresse des benötigten Unterprogramms. Auf diese Adresse wird der Programmzähler eingestellt, dessen Stand dann bei jedem Aufruf eines Befehls des Unterprogramms wiederum jeweils um eins erhöht wird.
Der letzte Befehl eines Unterprogramms ist der Befehl RET. Wird dieser Befehl gegeben, soll das Hauptprogramm mit der Befehlsspeicheradresse fortgesetzt werden, die als nächste derjenigen folgt, die den CALL-Befehl enthielt. Das ist aber genau derjenige Programmzählerstand, der nach Absenden der zum CALL-Befehl führenden Adresse gebildet wird.
Wir erinnern uns, daß der Programmzähler jeweils sogleich dann um eins erhöht wird, wenn sein bisheriger Stand zum Speicherplatzaufruf abgesendet ist. Diese sofortige Erhöhung wird eigens zu dem Zweck durchgeführt, daß man im Fall eines CALL-Befehls bereits die Rückkehradresse vorzuliegen hat. Man muß diese Adresse nur noch speichern. Dazu ist der STACK projektiert.
Mit dem CALL-Befehl wird also die nach Abarbeitung des Unterprogramms benötigte Rückkehradresse gespeichert. Der Befehl RET aktiviert dann diese Adresse wieder.

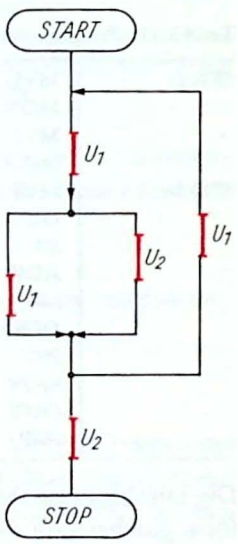


Bild 3.39
Schematische Darstellung eines Programmablaufplans mit Unterprogrammen

Vorbereitung zum Unterprogrammsprung, Abarbeitung des Unterprogramms und Rückkehr zum Hauptprogramm sind in den **Bildern 3.40 und 3.41** erläutert.
Zur Vereinfachung der Darstellung wollen wir in diesem Abschnitt übersehen, daß der CALL-Befehl wegen der Adressenübermittlung und der meist zu geringen Busbreite ein Mehrwortbefehl ist. Wir stellen also für den gesamten CALL-Befehl hier nur eine Speicheradresse dar.
Das auf den Speicherplätzen m bis $m + k$ abgespeicherte Unterprogramm wird zweimal aufgerufen. Der erste CALL-Befehl steht auf Platz 21, der zweite auf Platz 49. Nach Beendigung der Abarbeitung des Unterprogramms mit dem Befehl RET wird das Hauptprogramm mit dem jeweils nach dem CALL-Befehl folgenden Befehl (Befehl des Speicherplatzes 22 bzw. 50) fortgesetzt.
Zur Realisierung des Aufrufs eines Unterprogramms mit anschließender Rückkehr in das Hauptprogramm benötigt man einen STACK mit einem Register – so, wie es bei unserem einfachen Mikroprozessor projektiert war.
Es kann aber auch der Fall eintreten, daß innerhalb der Abarbeitung eines Unterprogramms noch ein nächstes Unterprogramm angefordert wird. Man spricht dann von *verschachtelten* Unterprogrammen. Solche Verschachtelungen können auch in mehrfacher Form auftreten. Dann muß der STACK entsprechend viele Register aufweisen.

In der gegenwärtigen Technik wird der STACK im RAM-Bereich definiert. Dazu wird im Mikroprozessor ein Register zur Aufbewahrung der jeweils aktuellen STACK-Adresse vorgesehen. Dieses Register bekommt den Namen *Stackpointer*(STACK-Zeiger). Erfolgt ein Unterprogrammaufruf, so wird der Stackpointerinhalt dekrementiert und bildet damit die Adresse desjenigen RAM-Platzes, auf dem nun der Programmzählerstand abgelegt wird.
Ist die Adreßbreite größer als die Speicherbreite, wird die Adresse schubweise übertragen und vor jedem Schub der Stackpointerinhalt dekrementiert.

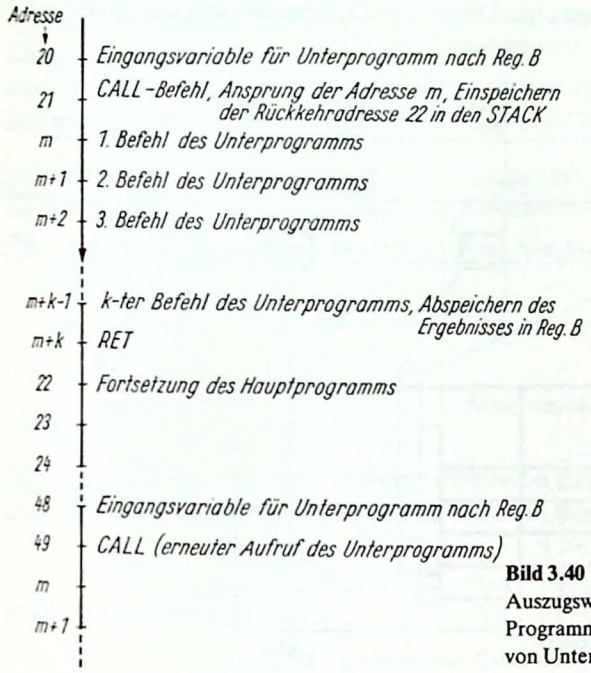


Bild 3.40
Auszugsweise Darstellung eines nach Einzelbefehlen detaillierten Programmablaufplans zur Erläuterung der Organisation von Unterprogrammabläufen

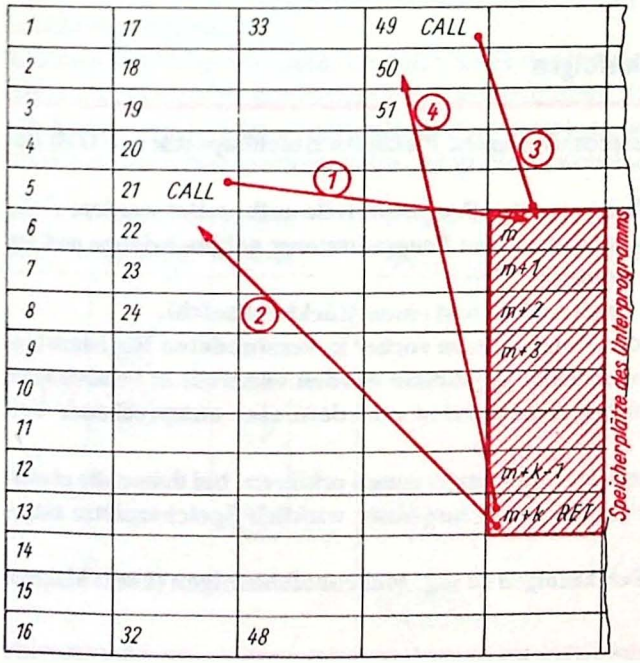


Bild 3.41
Erläuterung des Einschiebens eines Unterprogramms anhand eines Speicherplans

Im **Bild 3.42** ist dargestellt, wie sich der Stand des Programmzählers, der Inhalt des Stackpointers und des STACK bei einer Unterprogrammverschachtelung ändern (Annahme: Speicherbreite gleich Adreßbreite).
Beim RET-Befehl wird der Programmzähler mit dem Inhalt desjenigen RAM-Platzes geladen, der gerade durch den Stackpointer adressiert wird. Danach wird der Stackpointer inkrementiert.
Diese Art des Umspeicherns, wie sie beim STACK zum Aufbewahren der Rückkehradressen angewendet wird, nennt man das LIFO-Prinzip (LIFO ≙ last in/first out).
Das LIFO-Prinzip wird nicht nur für die Adressenspeicherung bei verschachtelten Unterprogrammen, sondern auch für andere Abspeicherungen, wie Daten und Operandenadressen sowie insbesondere bei Rettungsaktionen nach Interrupt, verwendet.

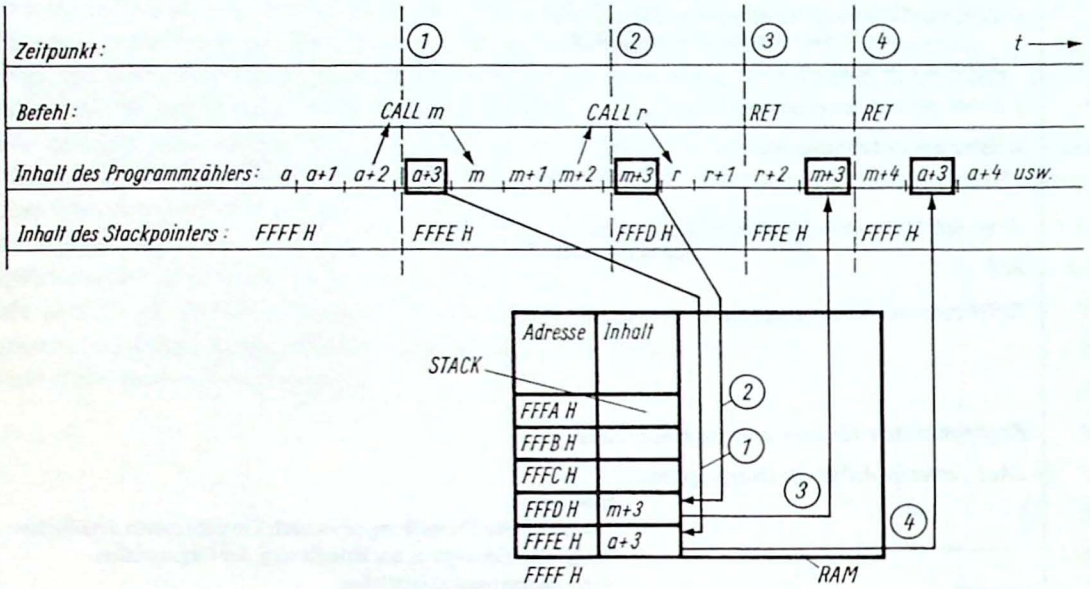


Bild 3.42. Veranschaulichung eines Programmablaufs bei Auftreten von verschachtelten Unterprogrammen

3.9.3. Definition von Makrobefehlsfolgen

Wir wissen: Mit Hilfe der Unterprogrammtechnik können Plätze im Befehlsspeicher (ROM) eingespart werden.
Zu Unterprogrammen können alle mehrfach benutzten Programmteile aufbereitet werden.
Vorteil: Alle Unterprogramme brauchen nur einmal vom Programmierer aufgeschrieben und nur einmal im Befehlsspeicher gespeichert zu werden.
Nachteile: Das Unterprogramm benötigt einen Aufruf- und einen Rückkehrbefehl.
Die vom Unterprogramm zu verarbeitenden Daten müssen vorher in verabredeten Registern hinterlegt werden. Die vom Unterprogramm erzielten Ergebnisse werden ebenfalls in verabredeten Speicherzellen übergeben. Diese zusätzlichen Arbeitsstufen erfordern eine entsprechende Verarbeitungszeit.
Man wird folglich nur solche Programmteile zu Unterprogrammen erklären, bei denen die zusätzlichen Zeitverzögerungen nicht ins Gewicht fallen und insgesamt wirklich Speicherplätze eingespart werden.
Man kann aber auch mehrfach benutzte Befehlsfolgen zu sog. Makrobefehlsfolgen (kurz: Makros) zusammenfassen.

Die Befehlsfolge eines Makros wird vom Programmierer festgelegt und mit einem Namen bezeichnet. Der Programmierer braucht diese Befehlsfolge ebenfalls nur einmal im Quellenprogramm in einer einzuhaltenden Form aufzuschreiben. An den Stellen, an denen diese Befehlsfolge jeweils im Programm erscheinen soll, ist dann nur noch der Name des Makros aufzuführen, und es sind gegebenenfalls benötigte Variable anzugeben. Beim Assemblieren dieses Programms sorgt dann der Rechner dafür, daß jeweils die volle, benötigte Befehlsfolge des Makros an diese Stelle geschrieben wird.

Mit diesem Verfahren wird also nur dem Programmierer die Arbeit erleichtert. Es werden keine Plätze im Befehlsspeicher eingespart, aber es gibt auch keine Zeitverzögerungen durch zusätzliche Befehle.

Zum Kennenlernen einer Makrodefinition wollen wir unser zweites kleines Beispiel von S. 174 als Makro definieren und ihm den Namen INA *geben* ***(Tafel 3.22).***

Name	Operation	Variable
INA	MACRO	
	IN	
	MOV	B,A
	MVI	A,9
	ADD	B
	OUT	
	ENDM	

Tafel 3.22
Beispiel einer Makrodefinition

Dazu wird das Programm in Assemblersprache aufgeschrieben und eine Definitionszeile voran- und eine Abschlußzeile nachgestellt. In unserem Beispiel lautet die vorangestellte Zeile:

INA MACRO

und die nachgestellte Zeile:

ENDM (soviel wie Ende des Makros).

Damit werden dem Übersetzungsrechner der Makroname sowie Anfang und Ende der Makrobefehlsfolge mitgeteilt.

Will dann der Programmierer an einer Stelle des Programms die Makrobefehlsfolge zur Wirkung bringen, so braucht er nur in die Operationsspalte den Namen INA zu schreiben **(Bild 3.43).**

Der Makro wirkt, vom Programmierer aus gesehen, wie ein neuer Befehl. Man kann also mit Makros das Befehlsrepertoire selbst vergrößern, wobei sich diese Erweiterung nur beim Schreiben

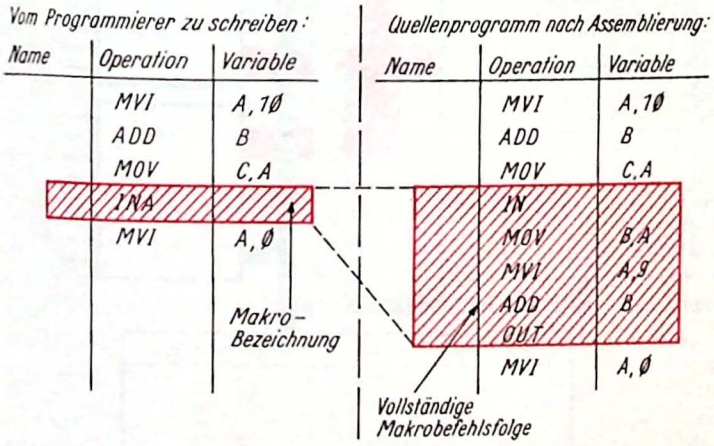


Bild 3.43. Erläuterung des Umsetzens eines Makroaufrufs bei der Assemblierung

des Quellenprogramms auswirkt. Leider muß man dann in jedem Programm die Makrodefinition erneut niederschreiben.
Wenn sich aber herausstellt, daß bestimmte Makros bei vielen Programmen benötigt werden, kann man noch einen Schritt weitergehen und diese Makros fest in das Übersetzungsprogramm aufnehmen. Dann entfällt die Notwendigkeit der jeweils erneuten Makrodefinition.
Abschließend sei noch bemerkt: Welche der beiden Techniken (Unterprogramm oder Makro) den jeweils größten Vorteil bietet, muß von Fall zu Fall überprüft werden.

3.10. Weiterentwicklung des Mikroprozessor/Mikrorechner-Konzepts

Fassen wir noch einmal zusammen:
Ein Mikroprozessor ist die zentrale Verarbeitungseinheit eines Mikrorechners. Er hat die Aufgabe, Datenübertragung und Datenbearbeitung im Sinn der einwirkenden Befehlsfolge zu organisieren. Da der Mikroprozessor nur Teil des Mikrorechners ist, kann sein Konzept nur im Zusammenhang mit der Gesamtkonzeption eines Mikrorechners entwickelt werden.

Wir wollen uns nun damit beschäftigen, wie das Mikroprozessor/Mikrorechner-Konzept weiterentwickelt wurde. Betrachten wir dazu das **Bild 3.44**, auf dem die wesentlichen Blöcke eines Mikrorechners dargestellt sind. Wir erkennen folgende Systemelemente:

- Bussystem: Adreß-, Daten-, Steuerbus
- Arithmetik-Logik-Einheit (ALU)
- Registerblock
- Speicherverwaltungseinheit
- Speicherblock
- Mikrosteuerwerk
- Zähler-Zeitgeber-Block (CTC)
- Eingabe-Ausgabe-Block (E/A).

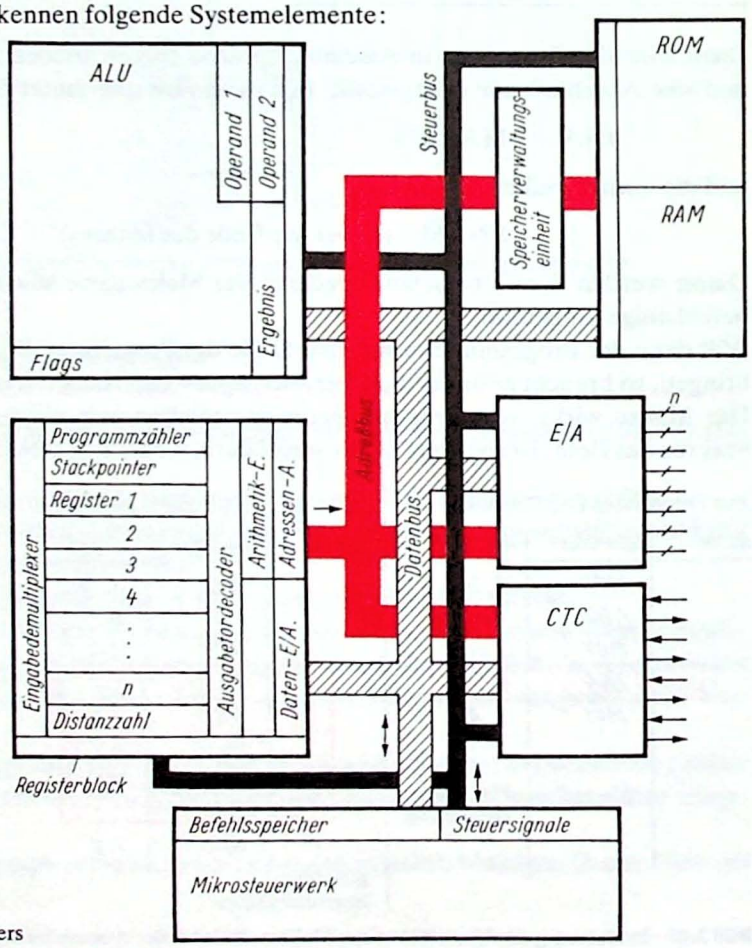


Bild 3.44. Blockbild eines Mikrorechners

Der zu einem Mikroprozessor gehörige und auf einem Chip gefertigte Teil besteht aus: Bussystem, ALU, Registerblock, Mikrosteuerwerk und manchmal auch noch Speicherverwaltungseinheit. Fertigt man auf diesem Chip noch eine bestimmte Kapazität an ROM-, RAM-Speichern sowie die Blöcke CTC und E/A hinzu, so spricht man von einem Einchipmikrorechner.
Da uns die grobe Ablaufsteuerung eines Mikrorechners bekannt ist, wollen wir uns mit den Detailkonzeptionen der einzelnen Blöcke befassen.

3.10.1. Bussystem

Das Bussystem hat die Aufgabe, Adressen, Daten und Steuersignale an die jeweiligen Empfänger weiterzuleiten.
Dabei werden oftmals (zur Einsparung von Leitungen) Adressen und Daten im Zeitmultiplexbetrieb nacheinander über den gleichen Bus übertragen. Dagegen müssen die Steuersignale als Zustandsinformationen für Zeit, Ort und Verwendungszweck parallel zu den Adressen und Daten vorhanden sein.

Man hat bisher Bussysteme für die parallele Übertragung ausgehend von 4 und 8 Bit über 16 Bit bis zu 32 Bit geschaffen.
Interessant ist, daß alle Varianten nach wie vor im Einsatz sind. Es ist sogar so, daß die einfachen 4-Bit-Einchipmikrorechnersysteme in der größten Stückzahl produziert werden, weil ihr Einsatz in vielen Steuerungen wirtschaftlich effektiv ist.
Ehe man den Schritt vom 16-Bit- zum 32-Bit-System machte, wurde noch ein Zwischenschritt eingeschoben: ein gemeinsamer 16-Bit-Daten/Adreßbus und weitere 7 Bit zur Auffächerung der Adressen in 128 Bereiche (Segmente).
Auf der 32-Bit-Ebene wurde eine zur Standardisierung vorgeschlagene Multi-Bus-Architektur entworfen, die für den Mehrrechnereinsatz konzipiert ist. Bei ihr werden Daten und Adressen im Multiplexverfahren übertragen. Es können aber auch 8, 16 oder 24 Bit breite Wörter übermittelt werden.
Mit steigender Busbreite können bei Daten die Genauigkeit und bei Adressen der adressierbare Speicherbereich erhöht werden. *Sehen wir uns dazu einmal die Tafel 3.23 an.*

Tafel 3.23. Adressierbarer Speicherumfang in Abhängigkeit von der Anzahl der Bit in der Adresse

Anzahl der Bit in der Adresse	Adressierbarer Speicherumfang	Möglichkeit der Speicherung von Buchseiten oder Büchern	
		A	B
8	256 Byte = 1/4 KByte	–	–
16	65536 Byte = 64 KByte	16 Seiten	–
24	16777216 Byte = 16 MByte	16 Bücher	1/4 Buch
32	4000 MByte = 4 GByte	4000 Bücher	68 Bücher

A ≙ Bücher mit Buchstaben und Strichbildern
B ≙ Bücher mit Raster-Farbbildern

In der **Tafel 3.23** ist neben dem adressierbaren Speicherumfang der Vergleich mit der dabei jeweils möglichen Speicherung von Buchseiten oder Büchern angegeben. In Spalte A sind Bücher mit Buchstaben und Strichzeichnungen erfaßt. Dabei wurde abgeschätzt, daß Buchstaben und Strichzeichnungselemente ungefähr die gleiche Speicherkapazität benötigen und daß auf eine Seite etwa 4000 Buchstaben kommen, die mit jeweils 1 Byte abgespeichert werden. Ein Buch wurde dabei aus 256 Druckseiten bestehend angesetzt.

In Spalte B sind Bücher mit Farbbilddarstellung im Rasterdruck aufgeführt. Hier wurde angenommen, daß ein Farbbild der Fläche 20 cm × 30 cm etwa 300000 Bildpunkte enthält und 8 Bit für die Helligkeitsinformation sowie 8 Bit für die Farbinformation angewendet werden. Ein Buch wurde aus 100 solcher Bilder bestehend angenommen. Textseiten sind in diesem Fall vernachlässigbar.
Vielleicht werden Ihnen durch die Buchvergleiche die adressierbaren Speicherumfänge verständlich.
Man braucht bei Ausnutzung dieser Möglichkeiten dann nur noch ein komfortables elektronisches Lesegerät (Farbdisplay) und ein entsprechendes Speichergerät, um sich eine entsprechend große Bibliothek auf kleinstem Raum zu halten.

3.10.2. Arithmetik-Logik-Einheit

Wir wissen: Die Arithmetik-Logik-Einheit hat die Aufgabe, Daten arithmetisch oder logisch in der vorgegebenen Datenbreite zu verknüpfen.
Das Ergebnis wird zum Ergebnisspeicher (Akkumulator) weitergeleitet, besondere Zustände des Ergebnisses werden im Flagregister abgespeichert.

Mit dem Größerwerden der Busbreite ist auch die mit einer Operation der ALU behandelbare *Datenwortbreite* angewachsen.
Während die ersten Mikroprozessoren nur die Operationen

- Addition/Subtraktion für Dualzahlen
- Links/Rechts-Rotation nur für Daten im Akkumulator
- Vergleich zweier Dualzahlen
- Logikoperationen: UND, ODER, EXOR

bei 8 Bit Datenbreite ausführten, sind später folgende Erweiterungen hinzugekommen, wobei sich diese auf die Daten in allen Register- und Hauptspeicherplätzen bei vergrößerter Wortbreite anwenden lassen:

- Addition/Subtraktion für BCD-Zahlen
- 16- oder 32-Bit-Multiplikation/Division
- spezielle Schiebebefehle
- Bit-Test- und Bit-Manipulationsbefehle.

Es wurden auch spezielle Arithmetik-Verarbeitungseinheiten entwickelt, die die vier Grundrechnungsarten bei Festkomma- oder Gleitkommadarstellung mit hoher Genauigkeit und in kurzer Zeit (1 bis 100 µs) durchführen.

3.10.3. Registerblock

Die Register haben die Aufgabe, Daten zeitweilig zu speichern, damit ein schneller Datentransfer zur ALU erfolgen kann.
Außerdem dienen sie zur Speicherung von Adressen und bestimmten Prozeßzuständen, z. B. für Flags.

In der ersten Zeit hat man zwischen daten- und adreßspeichernden Registern unterschieden. Vor allem deshalb, weil man den STACK zur Aufbewahrung von Programmfortsetzungsadressen gleich mit auf dem Prozessorchip integrierte.

Später hat man den STACK in den RAM-Bereich verlegt und nur noch den STACK-Adreßgeber (Stackpointer) im Mikroprozessor belassen.
Zur Adressierung des Datenbereichs im RAM wurden aber von Anfang an einige besonders bezeichnete Register aus dem Notizbuchspeicher als Adreßgeber verwendet. Als man später nahezu all diese Register als RAM-Adreßgeber zuließ, wurde dann die Unterscheidung zwischen Daten- und Adreßregistern aufgegeben. Man spricht daher heute von *Universalregistern.*
Aus diesen Bemerkungen ist zu entnehmen, daß offensichtlich die Adressierung einen bestimmenden Einfluß im Mikrorechnerbetrieb darstellt. Wir wollen uns daher mit diesem Problem etwas eingehender befassen. Betrachten wir hierzu die Bilder 3.44 und 3.45. Im **Bild 3.45** sind Einzelheiten eines Mikroprozessor-Registerblocks dargestellt.

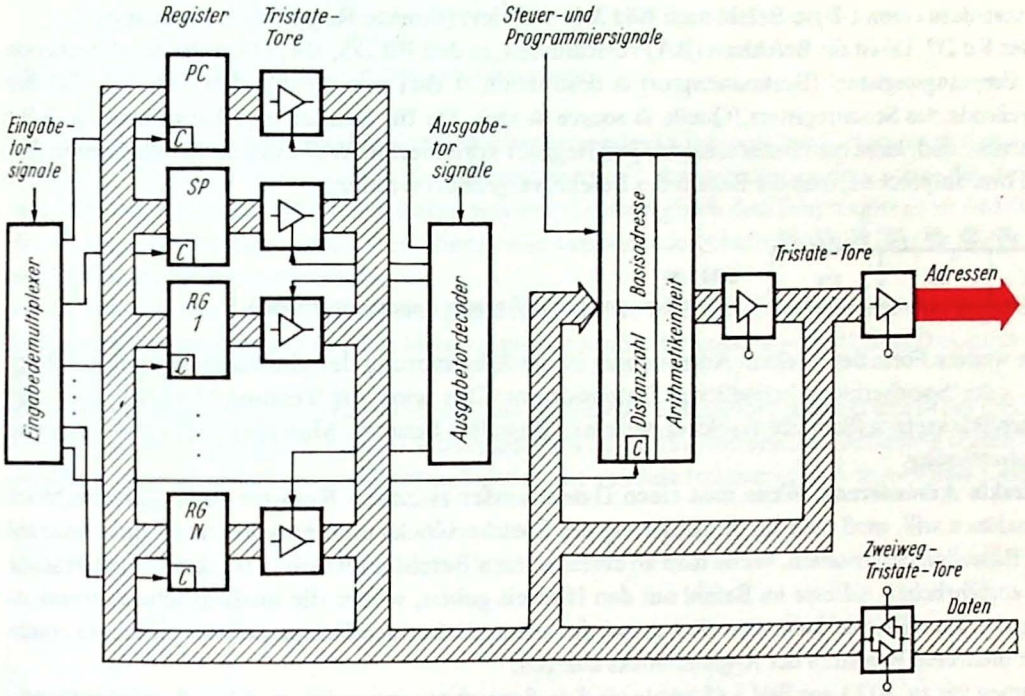


Bild 3.45. Registerblock eines Mikroprozessors/Mikrorechners

Der Registerblock besteht aus dem eigentlichen Registerteil mit Eingabedemultiplexer und Ausgabetordecoder sowie dem Arithmetikteil. Der Eingabedemultiplexer sorgt für den Übernahmeimpuls beim jeweiligen Daten einspeichernden Register. Der Ausgabetordecoder liefert das Toröffnungssignal für das jeweils als Sender zugelassene Register. Die Arithmetikeinheit dient zur Adreßberechnung aus Basisadresse und Distanzzahl. Ihr Ergebnis kann wieder in das jeweils die Adresse speichernde Register eingeschrieben, kann aber auch auf den Adreßbus ausgegeben werden.

Aufbau des Registerteils. Der Registerteil wird gebildet aus den beiden speziellen Registern Programmzähler (PC) und Stackpointer (SP) sowie den Universalregistern RG 1 bis RG N.
Der Programmzähler ist der erste und grundlegende Adreßgeber. Er dient zur Adreßbildung für die auf dem ROM in geordneter Reihenfolge abgespeicherten Befehle.
Der Stand des Programmzählers wird nach Absenden der Adresse für den zu holenden Befehl inkrementiert. Ist der zu holende Befehl ein *Sprungbefehl*, wird der Stand des Programmzählers neu eingestellt auf den Stand, den der Sprungbefehl fordert bzw. liefert.
Der Stackpointer ist der zweite Adreßgeber. Er verwaltet den STACK im RAM, indem er jeweils diejenige Adresse für den RAM gespeichert hat, von der aus die nächste Einkellerung oder Auskellerung beginnen kann.

Jedes der Universalregister kann Adressen oder Daten einspeichern und nach erfolgter Einspeicherung als Adreßgeber oder Datensender wirksam werden.

Welche Adressierungsarten gibt es nun, und wie wird die Adresse in den einzelnen Fällen bereitgestellt?

Direkte Adressierung. Bei der direkten Adressierung wird der zur Wirkung kommende Adreßanteil in vollständiger Form mit dem Befehl zusammen vom Programmspeicher geholt.

Beispiele hierfür bieten der JMP- und der CALL-Befehl (wie bei unserem einfachen Mikroprozessor).

Ein weiteres Beispiel hierzu bieten die Befehle zum Transport von Daten aus dem einen Register zu einem anderen Register. Sie werden möglichst als 1-Wort-Befehle aufgebaut. In der ersten Zeit hat man dazu einen 1-Byte-Befehl nach **Bild 3.46** definiert (direkte Registeradressierung).

In den Bit D7, D6 ist die Befehlsart (BA) verschlüsselt, in den Bit D5, D4, D3 steht der Adreßcode des Empfangsregisters (Bestimmungsort ≙ destination ≙ dst) und in den Bit D2, D1, D0 der Adreßcode des Senderegisters (Quelle ≙ source ≙ src). Da für Sender und Empfänger je 3 Bit reserviert sind, kann man damit maximal je 8 Register adressieren. Will man mehr Register in dieser Form ansprechen, muß die Bitzahl des Befehls vergrößert werden.

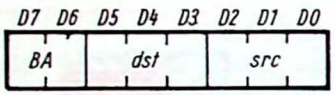


Bild 3.46
1-Byte-Transportbefehl (Register-Register-Transport)

Eine weitere Form der direkten Adressierung ist die Adressierung der einzelnen Bit von im Register- oder Speicherblock befindlichen Datenwörtern. Das wird zur Testung (Lesen) oder zum Setzen/Rücksetzen (Schreiben) solcher einzelnen Bitstellen benutzt. Man spricht hierbei auch von *Bitadressierung*.

Indirekte Adressierung. Wenn man einen Datentransfer zwischen Register- und Speicherblock vornehmen will, muß man zur Adressierung des Speicherblocks eine entsprechend hohe Anzahl von Bitstellen bereitstellen. Wenn man an einem kurzen Befehl interessiert ist, kann man anstelle der ausführlichen Adresse im Befehl nur den Hinweis geben, woher die ausführliche Adresse zu holen ist. Zum Beispiel kann man die Adresse des anzusprechenden Hauptspeicherplatzes in einem oder mehreren Registern des Registerblocks ablegen.

Nehmen wir an, RG 1 aus Bild 3.45 werde als Adreßspeicher verwendet, und RG 2 soll der Datenempfänger sein. Dann lautet der Befehl MOV RG 2, (RG 1) und wird, wie im **Bild 3.47** dargestellt, abgearbeitet.

- (RG 1) soll dabei bedeuten:
  Inhalt desjenigen Hauptspeicherplatzes, dessen Adresse im Register 1 gespeichert ist.
- Der gesamte Befehl sagt also aus:
  Im Register 1 steht die Adresse eines Hauptspeicherplatzes, dessen Inhalt zum Register 2 übertragen werden soll.

Der im Befehl gegebene Adreßhinweis (RG 1) dient also dazu, aus dem Register 1 die tatsächliche oder ausführliche Adresse zu holen.

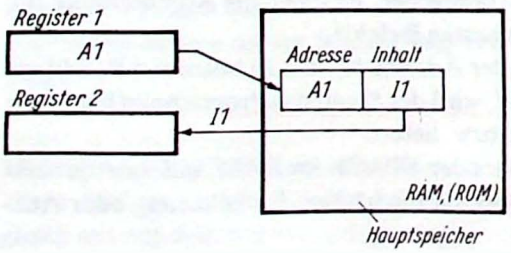


Bild 3.47
Veranschaulichung der indirekten Adressierung

Unmittelbare Adressierung. Bei der unmittelbaren Adressierung braucht man die Adresse der Quelle nicht anzugeben, da die Quelle ein vorausbestimmter Teil des Befehls ist, d. h., der Befehl liefert die zu transportierenden Daten **(Bild 3.48)**.

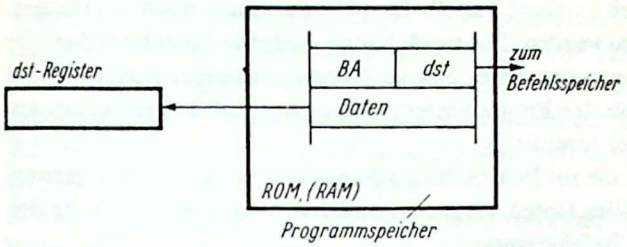


Bild 3.48. Veranschaulichung der unmittelbaren Adressierung

Der erste Teil des Befehls wird in den Befehlsspeicher des Mikrosteuerwerks transportiert und dort decodiert. Dadurch erkennt dann das Mikrosteuerwerk, daß mit dem zweiten Schub des Befehls die zu transportierenden Daten geliefert werden, die dann gleich dem Empfangsregister (dst-Register) übergeben werden. Diese Vorgehensweise entspricht der Abarbeitung des Befehls MVI A unseres einfachen Mikroprozessors.
Solche Befehle werden beispielsweise gebraucht, um den nach dem Einschalten des Mikrorechners noch unbestimmten Inhalt des RAM an vorgegebenen Stellen mit den gewünschten Daten aus dem ROM zu füllen.
Relative Adressierung. Hierbei wird mit dem Befehl eine Distanzzahl geliefert und in das Distanzzahlregister eingespeichert. Diese Distanzzahl wird im Verlauf der Befehlsabarbeitung vorzeichengemäß zu einer Basisadresse addiert und gibt so die absolute (tatsächliche) Adresse **(Bild 3.49)**.

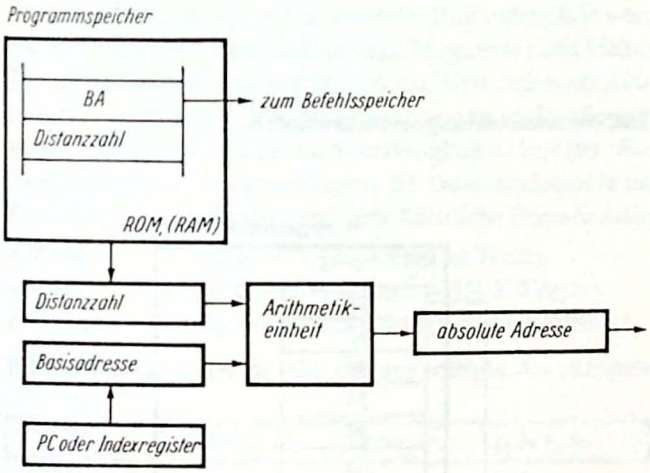


Bild 3.49. Veranschaulichung der relativen Adressierung

So kann man z. B. Sprungbefehle mit relativer Adressierung (relative Sprünge) definieren, die als Adreßteil nur die Distanzzahl liefern. Durch vorzeichenbehaftete Addition der Distanzzahl zum Stand des Programmzählers erhält man dann den neuen Stand des Programmzählers.
Hierzu muß der Stand des Programmzählers zum Operandenregister „Basisadresse" der Arithmetikeinheit übertragen werden (Bild 3.45). Die Arithmetikeinheit bildet dann die vorzeichenbehaftete Summe aus Basisadresse und Distanzzahl. Anschließend wird dann der Programmzähler (PC) mit dem Ergebnis geladen und erhält so einen sprunghaft veränderten Zählerstand.

Völlig gleichartig kann man auch mit Basisadressen arbeiten, die in einem Universalregister des Registerblocks abgelegt sind und Datenspeicherplätze im Speicherblock adressieren sollen. Das ist besonders vorteilhaft bei der Arbeit mit Datentabellen. In einem zu benennenden Register (meist als Indexregister bezeichnet) wird die Basisadresse der zu bearbeitenden Tabelle abgelegt.
In den Befehlen zur Bearbeitung von Daten in dieser Tabelle braucht dann nur noch die Distanzzahl zur abgelegten Basisadresse genannt zu werden. Die tatsächliche Adresse berechnet dann die Arithmetikeinheit des Registerblocks. Man nennt diese relative Adressierungsart auch *indizierte Adressierung*. Sie bringt Erleichterungen für den Programmierer, da er nur mit den kleineren und überschaubareren Distanzzahlen zu arbeiten braucht.

Blockweise Adressierung. Es gibt Befehle, die zur Bearbeitung von ganzen Datenblöcken verwendet werden. Ein solcher Befehl nimmt mit allen Daten des gesamten Blocks hintereinander die gleiche Manipulation vor. Beispiele für solche Befehle sind:

- Blocktransferbefehle
- Blocksuchbefehle
- Befehle für blockweise Eingabe/Ausgabe.

Beim Blocktransfer **(Bild 3.50)** werden die Daten eines Blocks der Blocklänge n in einen anderen Adressenbereich übertragen.

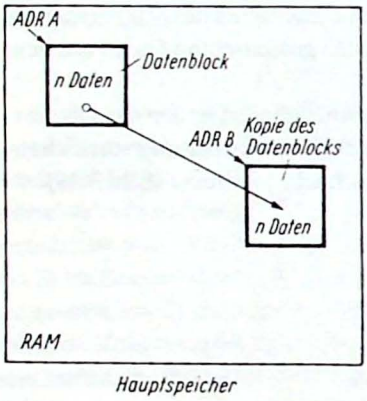


Bild 3.50. Veranschaulichung des Blocktransfers

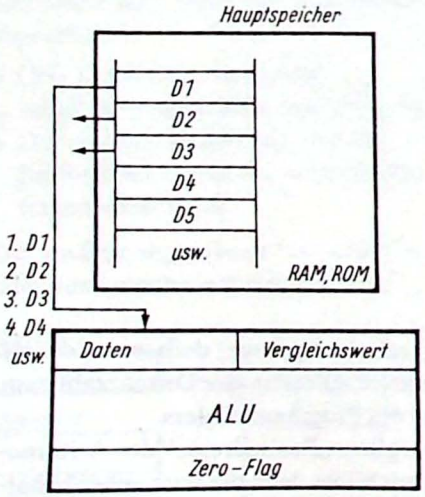


Bild 3.51. Veranschaulichung der Abarbeitung eines Blocksuchbefehls

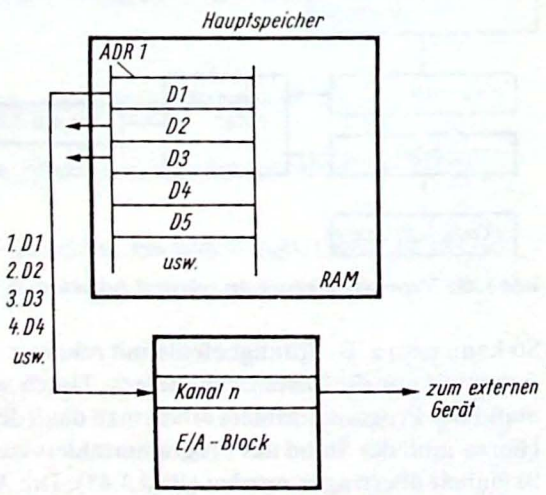


Bild 3.52. Veranschaulichung der blockweisen Eingabe/Ausgabe

Beim Blocksuchbefehl **(Bild 3.51)** wird der Reihe nach von jedem Datenwort eines Blocks eine Kopie in ein Operandenregister der ALU übertragen. In das andere Operandenregister der ALU kommt eine Kopie des gesuchten Vergleichsworts. Durch Differenzbildung wird auf Übereinstimmung beider Wörter geprüft. Liegt Übereinstimmung vor, werden das Zero-Flag gesetzt und die weitere Suche im Block eingestellt. Eine Abspeicherung des Ergebnisses im Akkumulator findet nicht statt. Der Inhalt des Akkumulators bleibt unverändert.
Nach Speicherung der Fundstellenadresse kann man durch Rücksprung zum Blocksuchbefehl die Suche nach dem Vergleichswort im noch nicht durchsuchten Blockteil fortsetzen lassen.
Bei der blockweisen Eingabe/Ausgabe findet ein Datentransfer über den E/A-Block zum oder vom Hauptspeicher (Speicherblock) statt.
Bei der Blockausgabe werden, beginnend von einer vorgegebenen Anfangsadresse ADR 1, der Reihe nach alle Speicherplatzinhalte des Datenblocks im Hauptspeicher über einen angewählten E/A-Kanal ausgegeben **(Bild 3.52)**.
Bei einer Blockeingabe werden die von einem externen Gerät über die E/A-Einheit gelieferten Daten, beginnend von der vorgegebenen Anfangsadresse, der Reihe nach im Hauptspeicher eingespeichert.

3.10.4. Speicherverwaltungseinheit/Speicherblock

Speicherverwaltungseinheit (MMU ≙ memory management unit) und Speicherblock sind informationstechnisch eng gekoppelt.
Die Speicherverwaltungseinheit sorgt für die Aufbereitung von Adressen (memory mapping) und dient zur Speicherüberwachung (memory management).

Diese Baugruppe ist erst in späterer Zeit entwickelt worden, als mikroelektronische Speicher mit größerer Speicherkapazität (einige Megabyte) und Mehrrechnersysteme zur Anwendung kamen.
Die *Adressenaufbereitung* dient dazu, Erleichterungen für den Programmierer und den Programmbenutzer zu bringen. Mit dem Übergang zu großen Speicherkapazitäten werden die Adreßangaben so unhandlich, daß es große Schwierigkeiten bereitet, die Übersicht zu behalten.
Deshalb hat man vorgeschlagen, die Gesamtadresse in mehrere Teilangaben aufzuteilen.
So besteht eine mögliche und gebräuchliche Form in folgender Einteilung:

- Zeile (1 Zeile = 1 gespeichertes Wort)
- Seite (1 Seite = 256 Zeilen = 1/4 KWörter)
- Segment (1 Segment = 256 Seiten = 64 KWörter).

Bild 3.53 zeigt einen Befehl, der aus solchen Adreßangaben aufgebaut ist.

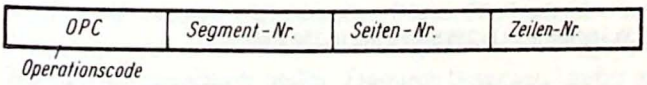


Bild 3.53. Befehl mit unterteilten Adreßangaben

Im **Bild 3.54** ist ein Speicherplan dargestellt, der diese Adreßangaben bei dichtester Packung demonstriert. Die in diesem Speicherplan schraffiert herausgehobene Zeile ist durch folgende Nummernangaben gekennzeichnet:

Segment-Nr.: 2
Seiten-Nr.: 2
Zeilen-Nr.: 4

Das ergibt folgende Kurzformdarstellung des Befehls:

OPC/ 2 / 2 / 4

In Binär- und Hexadezimalcodeform lautet die Adresse:

0000 0010 0000 0010 0000 0100 B = 020204 H

So einfach sieht die Adresse aber nur aus, wenn die Zeile 0 der nullten Seite vom nullten Segment auf der absoluten Adresse 0 steht und die Packung der Segmente so bleibt, wie sie im Bild 3.54 dargestellt ist. Will man aber eine beliebige Lage der einzelnen Segmente zulassen (reale Verhältnisse ergeben mitunter solche Forderungen), so ist es übersichtlicher, eine Tabelle mit den *Basisadreßwerten* der einzelnen Segmente anzulegen. Vom Basisadreßwert gelangt man zur Segmentanfangsadresse, indem man rechts 8 Nullen anfügt.

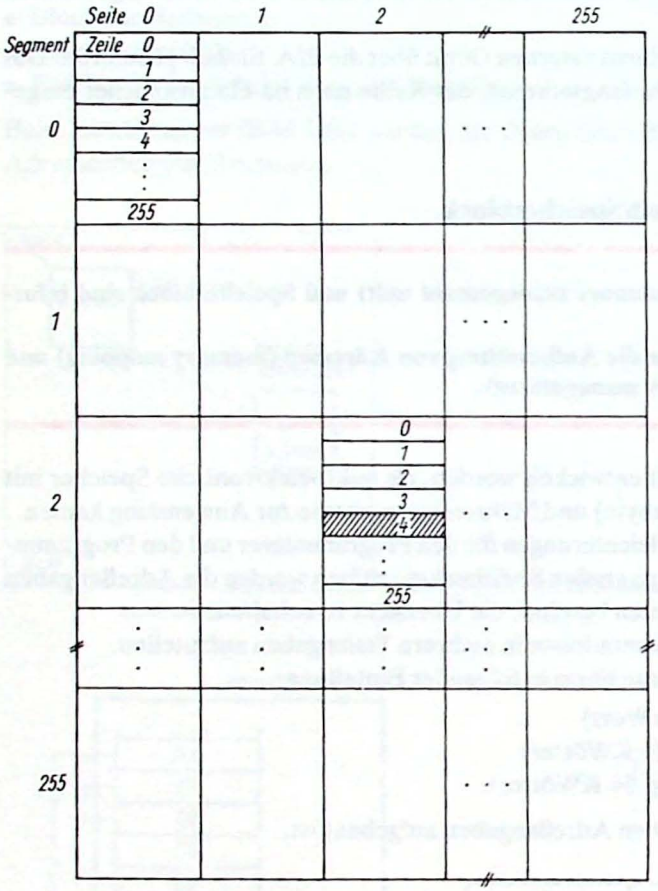


Bild 3.54. Plan zur Einteilung eines Speichers in 256 Segmente zu je 256 Seiten mit je 256 Zeilen

Wenn z. B. das von uns ausgewählte Segment Nr. 2 die Segmentanfangsadresse 057A00 H aufweist, so berechnet sich die absolute oder physikalische Adresse der von uns ausgewählten Speicherzelle zu:

physikalische Adresse = Segmentanfangsadresse + Seiten-Nr. × 2^8 + Zeilen-Nr.

	⌐Basisadreßwert¬	
Segmentanfangsadresse:	0000 0101 0111 1010 0000 0000	= 057A00 H
Seiten-Nr. × 2^8 + Zeilen-Nr.:	0000 0000 0000 0010 0000 0100	= 000204 H
physikalische Adresse:	0000 0101 0111 1100 0000 0100	= 057C04 H

Sicherlich sehen Sie jetzt den Unterschied in der Erfaßbarkeit der einzelnen Adreßformen ein.
Sprachlich werden folgende Ausdrücke benutzt:

Segment-Basis-Adresse = Segmentanfangsadresse
virtuelle Adresse = Adresse mit Angabe der Segment-, Seiten- und Zeilen-Nr.
physikalische Adresse = absolute oder reale Adresse

Das heißt: Der Mensch arbeitet mit der leicht überschaubaren virtuellen Adresse, und die Speicherverwaltungseinheit berechnet die reale, physikalische Adresse.

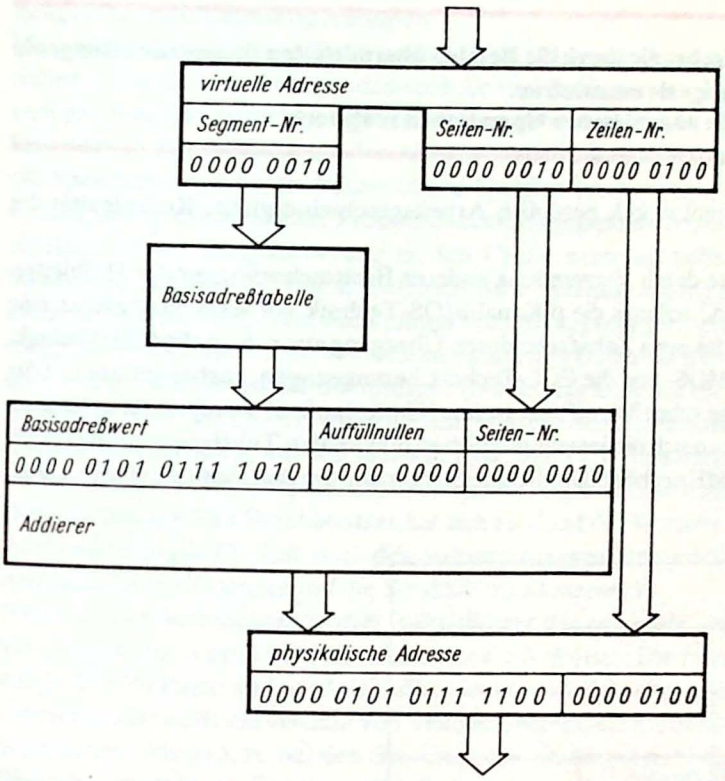


Bild 3.55. Veranschaulichung der Arbeitsweise des Adreßberechners einer Speicherverwaltungseinheit

Dazu benötigt die Speicherverwaltungseinheit einen Speicher, in dem die Basisadressen der einzelnen Segmente in nach Segmentnummern geordneter Reihenfolge stehen, und einen Addierer **(Bild 3.55)**.
Die Segment-Nr. führt auf eine 16-Bit-Angabe (Basisadreßwert), die die Basisadreßtabelle liefert. Diese 16 Bit stellen die beiden oberen Byte der 24-Bit-Segmentanfangsadresse dar. Die unteren 8 Bit dieser 24-Bit-Adresse sollen 0 sein. Das bedeutet, daß in diesem Fall die Zeilen-Nr. einer physikalischen Adresse auch bei beliebigen Verschiebungen der einzelnen Segmente im adressierbaren Bereich immer gleich bleibt. Dadurch brauchen hierbei auch nur die oberen 2 Byte der Segmentbasisadresse mit der durch Vornullen auf 16 Bit aufgefüllten Seiten-Nr. zusammenaddiert zu werden und ergeben dann die oberen 16 Bit der physikalischen Adresse. Die unteren 8 Bit der physikalischen Adresse ergeben sich aus der Zeilen-Nr.
Wie Sie jetzt erkennen können, bringt die Berechnung der physikalischen Adresse durch die Speicherverwaltungseinheit wesentliche Vorteile für die Speicherorganisation. Man kann einzelne Programmteile in Segmenten aufbauen, arbeitet dabei mit den Adreßteilen Segment, Seite, Zeile und kann später einem Speicherverwaltungsprogramm die effektive Einordnung der einzelnen Segmente in den physikalischen Adreßraum überlassen.

Neben der Adreßberechnung hat die Speicherverwaltungseinheit noch die Aufgabe, die Zugriffserlaubnis zu Speichereinheiten zu überwachen. Dazu hat sie eine Vergleichsschaltung, die Alarm auslöst, wenn die Befugnis zur Benutzung eines angesprochenen Speicherteils nicht gegeben ist. Ein solches Alarmsignal wird als Trap (Segment-Trap) bezeichnet. Dieses Trapsignal kann auch als CS-Sperre für den Speicher verwendet werden.

3.10.5. Mikrosteuerwerk

Das Mikrosteuerwerk hat die Aufgabe, die durch die Befehle übermittelten Steueranweisungen in zeitlich optimal ablaufende Steuersignale umzusetzen.
Durch die Steuersignale werden die angewiesenen Operationen realisiert.

Mikrosteuerwerke wurden weiterentwickelt bezüglich Arbeitsgeschwindigkeit, Komplexität des Befehlssatzes und der Struktur.
Die *Arbeitsgeschwindigkeit* konnte durch Verwendung anderer Basistechnologien der Halbleiterelemente erhöht werden. Während anfangs die p-Kanal-MOS-Technik vorherrschte, gab es eine Geschwindigkeitssteigerung um das etwa Zehnfache durch Übergang zur n-Kanal-MOS-Technik. Des weiteren wurden noch die CMOS- und die ECL-Technik herangezogen. Insbesondere mit der ECL-Technik werden zur Zeit die schnellsten Prozessoren (Gatterlaufzeit weniger als 1 Nanosekunde) gefertigt. Demgemäß haben sich die jeweils möglichen maximalen Taktfrequenzen von 250 kHz über 2,5 ... 8 MHz bis zu 25 MHz erhöht. Das ist eine Steigerung um den Faktor 100 gegenüber der Anfangsphase.

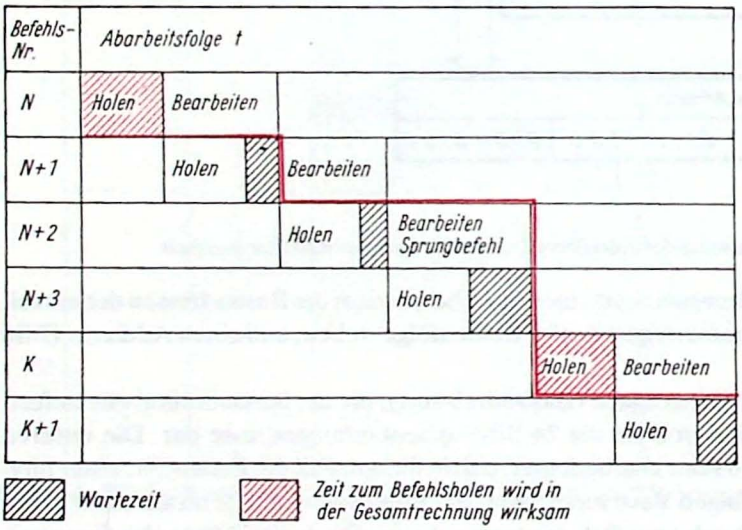


Bild 3.56. Prinzip des einfachen Pipelining-Verfahrens

Natürlich hat zur Erhöhung der Arbeitsgeschwindigkeit auch eine verbesserte Ablauforganisation bei gleichzeitiger Einflußnahme auf die Struktur des Mikrosteuerwerks beigetragen. Während man anfangs nach der Methode verfuhr, daß jeder geholte Befehl erst vollständig abgearbeitet wurde, ehe man den nächsten Befehl holte, arbeitet man heute meist nach dem sog. Pipelining-Verfahren **(Bild 3.56)**.
Danach wird die Abarbeitung der Befehle in zeitlicher Verschachtelung durchgeführt. Der erste Schritt auf diesem Weg bestand darin, parallel zur Bearbeitungsphase eines Befehls den nächsten

Befehl bereits zu holen. So ist im Bild 3.56 zu sehen, daß der Befehl $N + 1$ geholt wird, während sich der Befehl N in Bearbeitung befindet. Ist diese beendet, wird die Bearbeitung des Befehls $N + 1$ begonnen und der Befehl $N + 2$ geholt. Dieser Rhythmus wird nur dann unterbrochen, wenn ein Befehl (z. B. Sprung- oder CALL-Befehl) auftritt, der den Programmzähler neu einstellt. So wird die Zeit zum Befehlsholen in der Gesamtabrechnung größtenteils eliminiert.

Im nächsten Schritt wurden bisher rein sequentielle Abarbeitsfolgen darauf untersucht, ob nicht durch schubweise parallele Abarbeitung eine Beschleunigung erzielt werden könnte. Das läßt sich z.B. realisieren bei Multiplikation, Division, Mehrwortaddition und -subtraktion und bei zusammengesetzten Aritmetikoperationen.

Diese Beschleunigung gelingt natürlich nur dann, wenn die dazu benötigten Verarbeitungseinheiten auch parallel vorhanden sind. So sind Prozessoren bekannt, die über 6 Pipelining-Schritte verfügen und 6 Instruktionen gleichzeitig verarbeiten können.

Eine weitere Beschleunigung der Abarbeitung von Befehlsfolgen wurde durch Einführung des Cache-Speichers erreicht. In diesem Cache-Speicher (cache, engl., ist ein Versteck wertvoller Dinge) werden die in der nächsten Programmbearbeitungsphase benötigten Daten- und Programmteile abgelegt. Diese Einspeicherung in den Cache kann mit hoher Geschwindigkeit (gleichzeitige Schübe von mehreren Daten, Befehlen oder Befehlsgruppen) erfolgen. Dann befinden sich die nächsten Befehle im Cache und können von dort mit sehr geringem Adressieraufwand zur Bearbeitung abgeholt werden. Der Vorteil eines schnellen Datentransfers, den bisher nur die Registerspeicher boten, wird damit für die interne Abarbeitung auch auf die Befehle angewendet. Besonders vorteilhaft wird der Cache, wenn die eingespeicherten Befehlsfolgen oder Daten mehrmals hintereinander abgearbeitet werden müssen. Erst, wenn z. B. benötigte Daten nicht im Cache gefunden werden, wird der Hauptspeicher zur zeitaufwendigeren Übertragung einer Kopie aktiviert.

Die Komplexität des Befehlssatzes hat sich an Hand der Weiterentwicklung der Möglichkeiten der Arithmetik-Logik-Einheit und des Adressierungssystems erhöht. Gleichzeitig gab es dadurch ebenfalls Rückwirkungen auf die Struktur des Steuerwerks.

Waren in der Anfangsphase drei 8-Bit-Wörter der maximale Umfang eines Befehls (3-Byte-Befehl), so waren es etwas später schon vier 8-Bit-Wörter. Die Entwicklung ging weiter zu maximal vier 16-Bit-Wörtern und wird sicherlich damit noch nicht abgeschlossen sein.

Gewachsen ist auch die Anzahl von Teiloperationen, die maximal mit einem Befehl ausgelöst werden können, wie es z. B. bei den Blocktransfer- und Blocksuchbefehlen zu erkennen ist.

Weiterhin besteht die Tendenz, häufig benutzte, universelle Programmabschnitte fest im Steuerwerk zu implementieren. Zur Speicherung dieser Programmabschnitte können ROM-Speicher angewendet werden, die räumlich zum Mikrosteuerwerk gehören und von ihm bei Aufruf der speziellen Befehle jeweils auf dem entsprechenden Teilabschnitt abgearbeitet werden.

In Weiterverfolgung dieser Idee sind sog. Interpreter entwickelt worden, die Befehle in höherer Programmiersprache direkt verarbeiten.

Die gesamte Entwicklung strebt dahin, die Fähigkeiten der Mikroprozessoren bezüglich Arbeitsgeschwindigkeit, möglicher Genauigkeit und Vielseitigkeit zu erhöhen.

3.10.6. Zähler/Zeitgeber-Block (CTC)

Der Zähler/Zeitgeber-Block (CTC ≙ counter/timer-circuit) hat die Aufgabe, Zählvorgänge von elektrischen Impulsen zur Bestimmung einer Impulsanzahl oder einer definierten Zeitspanne selbständig durchzuführen.

Damit wird der Mikrorechner von diesem Teilprozeß entlastet und braucht sich nur zu gegebener Zeit mit dem Ergebnis der CTC zu beschäftigen.

Der Zähler/Zeitgeber-Block ist softwaremäßig programmierbar in bezug auf:

- Betriebsweise der einzelnen Kanäle
- Bedingungen für die Fertigmeldung jedes Kanals
- Interruptbeteiligung der Kanäle
- Interrupt-Adreß-Komponente.

Bild 3.57 zeigt einen CTC-Block mit vier Kanälen. Es ist zu erkennen, daß die CTC über alle drei Busse (Daten, Adressen, Steuersignale) mit dem Rechnersystem verbunden ist. Außerdem wirkt noch der Systemtakt Φ ein.

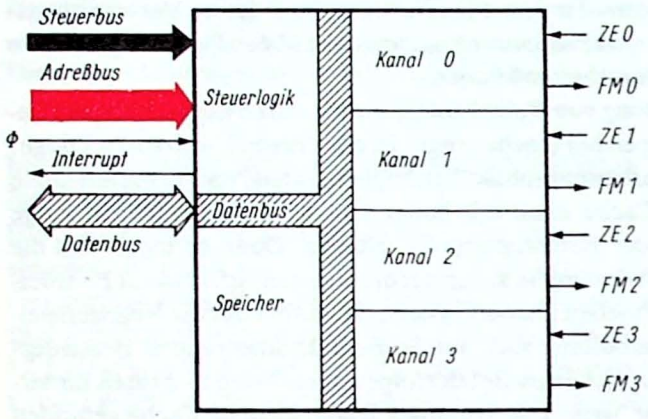


Bild 3.57. Grobe Darstellung eines CTC-Blocks mit 4 Kanälen
ZE ≙ Zähleingang; FM ≙ Ausgang für Fertigmeldung

Ist in einem Kanal ein vorprogrammiertes Zähl- oder Zeitergebnis erreicht, gibt dieser einen Fertigmeldeimpuls über den zugehörigen Ausgang FM ab. Hat dieser Kanal Interrupterlaubnis, so wird noch der Interruptausgang aktiv.
Auch bei dieser Baugruppe hat sich eine rasche Entwicklung vollzogen. Während sie in der ersten Phase noch nicht existierte, wurde sie über einige Entwicklungsetappen zu einem Baustein mit eigenem Steuerrechner und großem Komfort weiterentwickelt.
Doch betrachten wir uns zunächst das Prinzip. Auf der groben Blockdarstellung eines CTC-Kanals im **Bild 3.58** erkennen Sie als Grundelemente einen Zähler (CT) und ein Register (RG τ). Der Zähler arbeitet im Rückwärtszählbetrieb (Dekrementieren). Die Zählimpulse werden ihm über einen Multiplexer zugeführt. Dessen Steuersignal *e* entscheidet, ob diese Impulse vom Systemtakt Φ oder von einer am Eingang ZE1 angeschlossenen externen Impulsquelle herrühren.
Dadurch sind auch die beiden grundsätzlichen Betriebsarten definiert:

- Zählerbetrieb – die Zählimpulse kommen vom Eingang ZE
- Zeitgeberbetrieb – die Zählimpulse kommen vom Systemtakt.

Da die Frequenz des Systemtakts bekannt und wegen ihrer Quarzstabilisierung sehr genau ist, kann man aus dieser Frequenz über eine Impulszählung sehr exakte Zeitabschnitte gewinnen.
Der Zählvorgang verläuft i. allg. periodisch. Eine Zählperiode beginnt damit, daß der Zähler mit dem im Register gespeicherten Voreinstellwert (Zeitkonstante τ) geladen wird. Dann wirken die Zählimpulse solange ein, bis der Voreinstellwert zu 0 dekrementiert ist. Der Zähler gibt dann an seinem Ausgang ü einen Impuls ab, der als Fertigmeldung am Ausgang FM1 erscheint. Gleichzeitig wirkt dieser Impuls auch intern, so daß er die Neuladung des Zählers mit der Zeitkonstanten τ bewirken kann. Danach wird der gesamte Zählvorgang wiederholt.
Aus dem Fertigmeldeimpuls kann auch eine Interruptanmeldung beim Rechner ausgelöst werden.
Das Zeitkonstantenregister (RG τ) wird in einer vorbereitenden Initialisierungsphase durch Soft-

waresteuerung geladen. Das Tristate-Tor dient zur E/A-Entkopplung des Datenbusses. Durch einen entsprechenden Befehl kann der Rechner per Software den Auftrag geben, daß die Steuerlogik das Tristate-Tor öffnet und damit den Zählerstand auf den Datenbus schaltet. Dadurch kann dieser Stand vom Rechner gelesen werden.

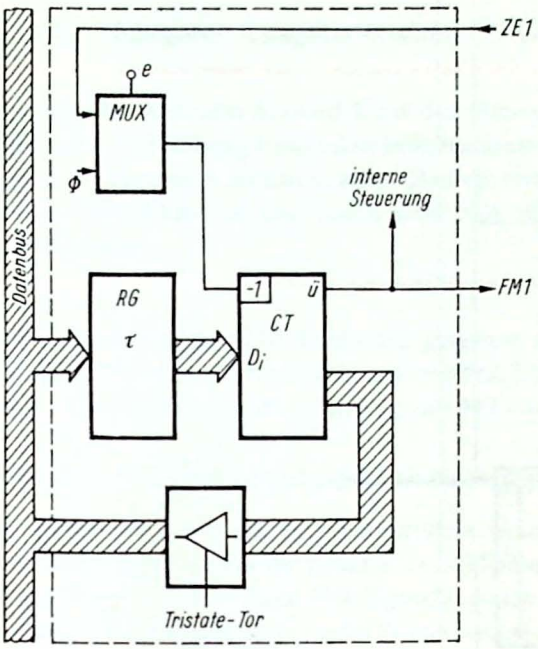


Bild 3.58. Grobe Blockbilddarstellung eines CTC-Kanals (Kanal 1)

Bild 3.59 zeigt ein vereinfachtes Signalflußbild eines CTC-Kanals. Die Lage der bisher bekannt gewordenen Elemente Register, Zähler, Eingangssignal-Multiplexer, Tristate-Tor ist geblieben, so daß die Orientierung nicht schwerfallen dürfte.

Neu hinzugekommen sind vier taktflankengesteuerte D-Trigger (G1, G2, G3, G4), drei UND-Gatter-Tore (G5, G6, G8) und ein Verzögerungsglied (G7). *Mit dieser Darstellung wollen wir uns noch ein etwas tieferes Verständnis erwerben.*

Die D-Trigger (G1, G2, G3) dienen zum Einspeichern der Programmierungswünsche. Sie werden in der Initialisierungsphase gesetzt oder rückgesetzt. Ist z. B. Trigger G1 gesetzt, so ist das Tor G8 geöffnet, und der beim Nulldurchgang des Zählers am Ausgang ü auftretende Fertigmeldeimpuls wird zur Interruptanmeldung durchgelassen (Interrupt enable). Ist G1 rückgesetzt, so kann kein Interrupt ausgelöst werden (Interrupt disable).

Trigger G2 wirkt auf Tor G5 ein. Ist der Trigger gesetzt, können die Zählimpulse wirksam werden. Ist er rückgesetzt, werden die Zählimpulse nicht durchgelassen. So können durch Softwaresteuerung Start und Stop ausgelöst werden.

Trigger G3 liefert das Steuersignal des Multiplexers, so daß je nach seinem Speicherzustand (0 oder 1) die Zählimpulse aus dem ZE1-Signal oder dem Systemtakt Φ ausgewählt werden.

Trigger G4 dient zur Synchronisierung der ZE1-Zählimpulse mit dem Systemtakt.

Die Glieder G6 und G7 dienen zur Realisierung der Neuladung des Zählers mit der Zeitkonstanten τ bei Erscheinen des Fertigmeldeimpulses. Durch G7 wird dieser Impuls zeitlich etwas verzögert, so daß durch das Neuladen der Fertigmeldeimpuls nicht verkürzt wird. Über das Steuersignal e_4 kann das Neuladen gestattet oder verhindert werden.

Die Wortbreite der Zeitkonstanten τ und des Zählers wird i. allg. zu 16 oder 32 Bit gewählt. Bei einer Wortbreite von 32 Bit beträgt der maximale Zählumfang etwa $4295000000 \approx 4{,}3 \cdot 10^9$ Im-

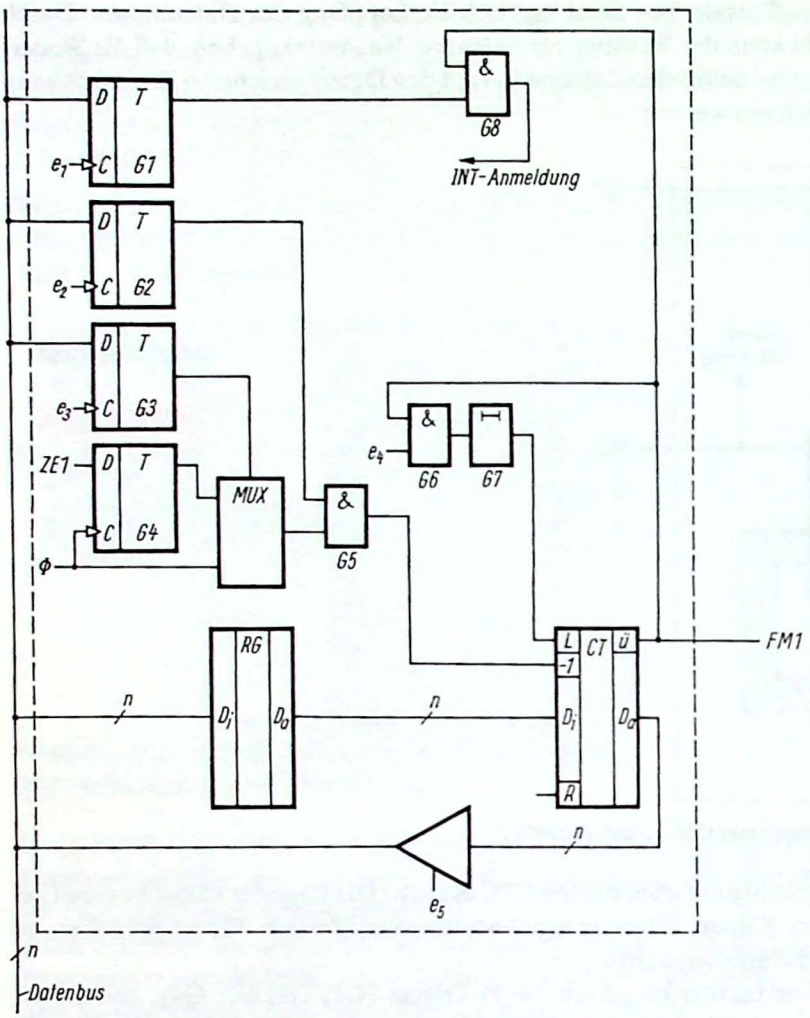


Bild 3.59. Vereinfachte Darstellung des Signalflußbilds eines CTC-Kanals (Kanal 1)

pulse. Treten die Impulse mit einer Frequenz von 1 MHz auf, so benötigt der Zähler dazu eine Zeitdauer von etwa 4295 Sekunden ≈ 1 Stunde und 11 Minuten (und das bei einer Einzelschrittauflösung von einer Mikrosekunde!).

Will man den Zählumfang noch weiter vergrößern, kann man die Kanäle kaskadieren. So zeigt **Bild 3.60** die Hintereinanderschaltung der Kanäle 1 und 2.

Bei den früher entwickelten CTC-Blöcken der 8-Bit-Technik hat man innerhalb eines Kanals zwei 8-Bit-Zähler kaskadiert und kam so auch auf die für einen Kanal annehmbare Wortbreite von 16 Bit.

Der Entwicklungstrend besteht darin, die CTC-Blöcke immer komfortabler und selbständiger zu gestalten. So gibt es heute CTC-Blöcke, die aus mehr als 10000 Transistoren bestehen und eine eigene Mikrorechnersteuerung haben.

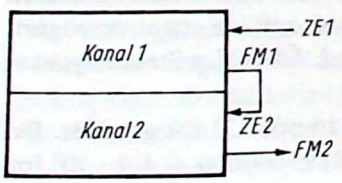


Bild 3.60
Blockbilddarstellung der Kaskadierung zweiter CTC-Kanäle

So sind die Zählereinheiten auf Vor- und Rückwärtszählbetrieb programmierbar. So können echte Tageszeitmeldepunkte vorgegeben werden (in Stunden, Minuten, Sekunden, Zehntelsekunden). So kann ein Stopp-Uhr-Betrieb realisiert werden. So kann das Zählergebnis nicht nur im Binärcode, sondern auch im BCD-Code oder als Uhrzeit ausgegeben werden.

3.10.7. Eingabe-Ausgabe-Einheit (E/A-Block)

Die Eingabe-Ausgabe-Einheit führt den Datenaustausch zwischen einem Mikrorechner und den mit ihm in Verbindung stehenden informationsverarbeitenden Baugruppen aus.
Solche Baugruppen können sein: Bedieneinrichtungen, Sensoren, ADU, DAU, Stellglieder, Anzeigeeinrichtungen, aber auch nicht zum Mikrorechner gehörige Speicher oder sogar andere Mikrorechner.

Die Entwicklung der E/A-Einheiten ging vom einfachen Pufferregister zur Zwischenspeicherung eines E/A-Wortes über schon kompliziertere E/A-IC mit eigener Steuerlogik bis hin zu intelligenten Verbindungsnetzwerken mit eigener Mikrorechnersteuerung.

3.10.7.1. Eingabe-Ausgabe-Einheit für serielle Datenübertragung

Die einfachste Form der E/A-Schnittstelle ist im **Bild 3.61** dargestellt. Dort sehen Sie nur eine in zwei Richtungen wirkende Tristate-Torschaltung für 1 Bit (bitserielle Übertragung).
Ist das Tor 1 geöffnet (und Tor 2 geschlossen), so übernimmt der Mikrorechnerdatenbus auf der Leitung 0 die Bitbelegung (1 oder 0) von der externen Baugruppe. Ist Tor 2 geöffnet (und Tor 1 geschlossen), so wird die Übertragungsrichtung umgekehrt. Der Mikrorechnerdatenbus ist dann der Sender.

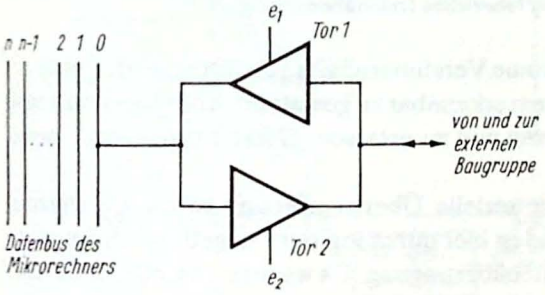


Bild 3.61
In zwei Richtungen wirkende Tristate-Torschaltung für 1 Bit (bitserielle Übertragung)

In dieser Darstellung gibt es folgendes Problem zu bedenken: Der Mikrorechnerbus wird mit seinen vielen Leitungen überhaupt nicht richtig genutzt, da nur eine einzige Leitung die Information trägt.
Das kann aber wie folgt verbessert werden: Durch Einfügen je eines Schieberegisters für beide Übertragungsrichtungen kann man die Belegungen so vieler, nacheinanderfolgender Bitstellen einspeichern, wie auf einmal vom Mikrorechnerbussystem parallel übertragen werden können **(Bild 3.62)**.
Ist z. B. nach der seriellen Eingabe von 8 Bit das Empfangsschieberegister voll, dann können die jetzt für jede einzelne Busleitung benötigten Tristate-Tore 1 geöffnet werden. Das Empfangsschieberegister, das die Serien-Parallel-Wandlung durchgeführt hat, wirkt dann als Sender und belegt mit seinen Parallelausgängen den Mikrorechnerdatenbus.
Bei der Ausgabe werden 8 Bit parallel in das Sendeschieberegister gegeben und anschließend nach Öffnen des Tristate-Tores 2 seriell mit Hilfe des Ausgabetaktes an die externe Baugruppe übergeben.

Verfügt auch die externe Baugruppe über einen Bus, so kann auch bei ihr eine solche Einrichtung zur Serien-Parallel- und Parallel-Serien-Wandlung und damit zur effektiveren Busauslastung eingesetzt werden.
Mit Hilfe dieses Wandlersystems bleibt der Datenbus längere Zeit für andere Datenübertragungen frei.
Für solche seriellen E/A-Schnittstellen sind universelle IC entwickelt worden, die neben der Parallel-Serien- und Serien-Parallel-Wandlung auch noch zahlreiche Kontrollfunktionen beim Empfang und Erkennungssignal-Generierungen beim Senden realisieren.

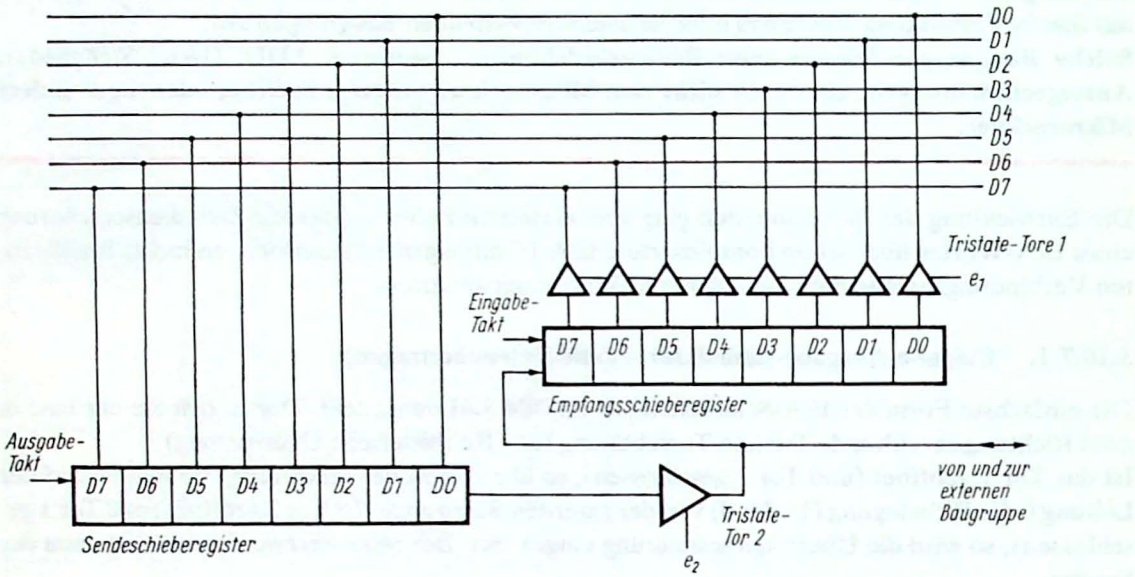


Bild 3.62. Serien-Parallel- und Parallel-Serien-Wandler bei der bitseriellen Datenübertragung

Dazu muß gesagt werden, daß für serielle Datenströme Vereinbarungen getroffen werden müssen, um Anfang und Ende des Datenstroms für das System erkennbar zu gestalten. Außerdem trifft man Maßnahmen, um Übertragungsfehler zu minimieren und zu erfassen. Diese Festlegungen nennt man ein *Datenprotokoll.*

Asynchrone, serielle Übertragung. Die einfachste serielle Übertragungsart ist die asynchrone, serielle Übertragung **(Bild 3.63)**. Der Wartezustand ist hier durch logisch 1 gegeben. Gibt es einen Wechsel nach logisch 0, so gilt das als Start der Datenübertragung. Es werden anschließend z. B. 7 gültige Datenbit übertragen. Dann folgt das Paritätsbit. Dieses ist 0, wenn die Anzahl der mit logisch 1 belegten Bit (D0 bis D6) ungerade war. Sonst wird das Paritätsbit auf 1 gesetzt.
Im Bild 3.63 lautet die zu übertragende Information: 23 H. Anschließend an das Paritätsbit folgen ein oder zwei Stoppbit, die durch den Zustand logisch 1 ausgedrückt werden. Dann kann erst das nächste Wort folgen.
Da jedes zu übertragende Byte mit Anfangs- und Endekennung versehen werden muß, ist diese Übertragungsart wenig effektiv. Man hat daher die synchrone, serielle Übertragung eingeführt.

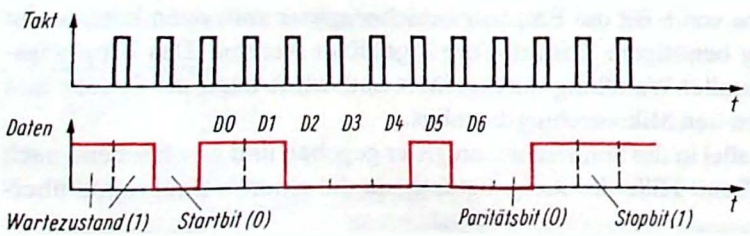


Bild 3.63
Zeitdiagramme für die asynchrone, serielle Übertragung

Synchrone, serielle Übertragung. Bei dieser stellt man jedem zu übertragenden Block ein oder zwei Synchronbyte voran und fügt am Ende zwei CRC-Byte (CRC ≙ cyclic redundancy control ≙ zyklische Redundanzkontrolle) hinzu. Die CRC-Byte werden aus einer Kontrollfunktion (Polynom) an Hand aller zur Übertragung gelangenden Datenbyte eines Datenblocks berechnet **(Bild 3.64)**.

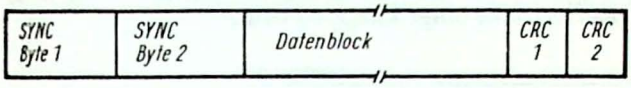


Bild 3.64. Aufbau des Informationsblocks bei der synchronen, seriellen Übertragung

Bei solchen seriellen, synchronen Schnittstellen wird eine Eingangskontrolle beim Empfang vorgenommen, um festzustellen, ob der Anfangssynchroncode schon eingetroffen ist. Erst nach Erkennung dieses Codes wird der Empfang des Datenblocks eingeleitet. Dabei wird erneut die Berechnung des CRC-Polynoms durchgeführt und das Ergebnis dann mit den CRC-Byte am Ende der Blockübertragung verglichen, bei Nichtübereinstimmung wird eine Fehlermeldung abgegeben. Ein grobes Blockbild für eine solche serielle Schnittstelle zeigt **Bild 3.65**.

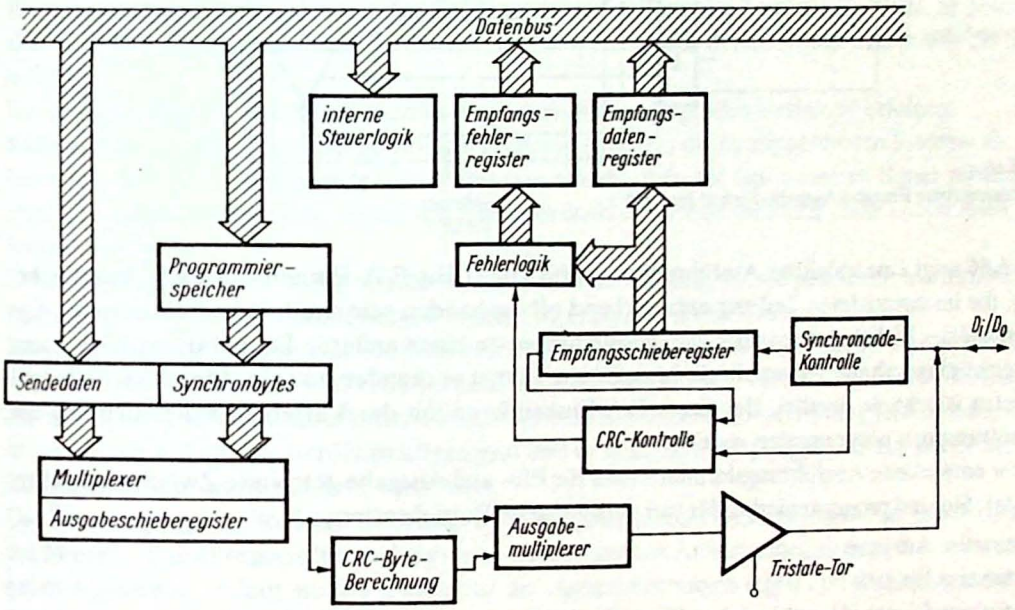


Bild 3.65. Vereinfachtes Blockbild einer synchronen, seriellen E/A-Schnittstelle

In der Initialisierungsphase müssen die Synchronbyte, das CRC-Polynom und andere Vorgabewerte durch Softwaresteuerung programmiert werden. Dann können die Startsignale für Senden oder Empfangen gegeben werden. Im Sendebetrieb wird mit den Synchronbyte begonnen. Dann folgt der Datenblock, bei dessen Übertragung die CRC-Berechnung läuft. Am Ende des Datenblocks werden die beiden Ergebnisbyte der CRC-Berechnung angefügt.

Für serielle E/A-Schnittstellen werden folgende Begriffe benutzt:

UART universal asynchronous receiver/transmitter ≙ universeller asynchroner Empfänger/Sender

USART universal synchronous/asynchronous receiver/transmitter

SIO serial input/output ≙ serieller E/A-Schaltkreis

3.10.7.2. Eingabe-Ausgabe-Einheit für parallele Datenübertragung

Hierzu zunächst eine Präzisierung der in knapper Form gefaßten Überschrift:
Eine vollständig parallele Übertragung größerer Informationsmengen ist nicht realisierbar. Es kann sich also nur um einzelne parallele Datenschübe handeln. Daher müßte die Überschrift exakt heißen: E/A-Einheit für bitparallele, wort- oder byteserielle Datenübertragung. Da dieser Ausdruck aber recht unhandlich ist, verwendet man meist die obige knappe Form.

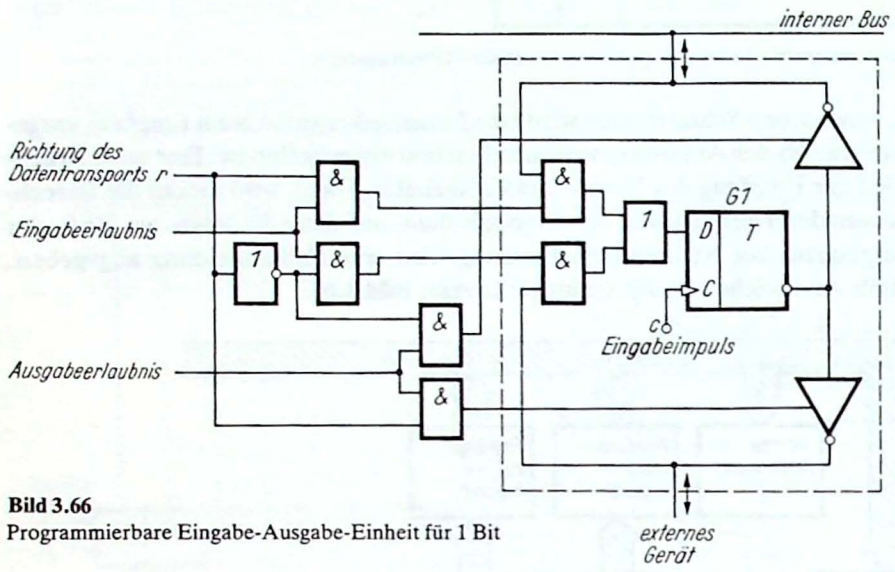


Bild 3.66
Programmierbare Eingabe-Ausgabe-Einheit für 1 Bit

Bild 3.66 zeigt eine mögliche Ausführungsform für eine 1-Bit-E/A-Einheit mit Zwischenspeicherung, die im umrandeten Teil nur entsprechend oft vorhanden sein muß, damit die gleichzeitige Übergabe der Bitbelegungen aller Datenbusleitungen an einen anderen Datenbus erfolgen kann. Die nicht eingerahmte Auswahllogik braucht nur einmal vorhanden zu sein. Sie steuert alle umrandeten Blockteile parallel. Bei dieser E/A-Einheit kann mit der Variablen *r* die Richtung des Datentransports programmiert werden.
Weiter entwickelte Ausführungsformen haben für Ein- und Ausgabe getrennte Zwischenspeicherregister. Sie sind programmierbar für vier verschiedene Betriebsarten:

- byteweise Ausgabe
- byteweise Eingabe
- byteweise Eingabe/Ausgabe (abwechselnder Betrieb in beiden Richtungen)
- Bit-Eingabe/Ausgabe (jeder einzelne E/A-Pin kann wahlweise als Eingangs- oder Ausgangspin programmiert werden)

Solche Baugruppen sind unter der Bezeichnung PIO (parallel input/output) bekannt geworden.
Weitere Mikrorechner-Zusatzschaltkreise für E/A-Operationen sind:
Direkter Speicherzugriffs-Baustein. Zum schnelleren Umladen von Daten von einem Speicher in einen anderen und zur schnellen Suche nach vorgegebenen Daten wurden spezielle IC für den direkten Speicherzugriff, sog. DMA-Bausteine (DMA ≙ direct memory access), entwickelt.
Mit ihnen ist ein wesentlich schnellerer Datentransfer möglich, da der Umweg der Daten über ein internes Register der CPU und das ständige Befehlsholen entfallen.
Speicher nach dem Schieberegisterprinzip. Solche Speicher heißen SAM (serial addressed memory) und dienen als Pufferspeicher zur schnellen Eingabe/Ausgabe von Datenblöcken. Die eingespeicherte Information kann als *Warteschlange* betrachtet werden, die zur Anpassung zwischen schnellen und langsameren Baugruppen zwischengespeichert wird.

Die Information, die als erste hineingebracht wird, geht als erste wieder hinaus. Daher heißt eine solche Baugruppe auch FIFO-Speicher (FIFO ≙ first in/first out).

Analogsignal-E/A-Baugruppe. Hierzu werden Analog-Digital-Umsetzer und Digital-Analog-Umsetzer mit in den E/A-Baustein integriert.

Auch bei den E/A-Bausteinen besteht der Trend, durch Anwendung eigener Steuerrechner zur Erhöhung der Variabilität des E/A-Blocks und zur Entlastung des Hauptrechners beizutragen.

3.10.7.3. Steuerung der Eingabe-Ausgabe-Prozesse

Bisher haben wir nur die Form des Datenflusses betrachtet: rein serieller oder bitparalleler, wortserieller Datenstrom. Bisher unberücksichtigt blieb die Frage, welcher Baustein jeweils die Erlaubnis zum Öffnen der betreffenden Tore erhält und die Ausgabe des Übernahmeimpulses vornimmt.

Diese Problematik ist von weitreichender Bedeutung und hat großen Einfluß auf die Effektivität eines verzweigten Informationsverarbeitungssystems. Bereits im Abschnitt 1.4.5. haben wir uns einführend mit dieser Problematik beschäftigt. Danach können zusammenarbeitende I-Systeme *hierarchisch* oder *gleichberechtigt* verbunden sein.

Beim hierarchischen System hat das übergeordnete I-System die Kommandogewalt. Es beschafft sich von den untergeordneten Systemen die Informationen, wertet sie aus und erteilt die Stellbefehle.

Die Beschaffung der Informationen kann nach zwei grundlegenden Verfahren erfolgen:

Polling. Hierbei fragt das übergeordnete I-System (Master) die untergeordneten Systeme ab. Das bedeutet, daß die untergeordneten I-Systeme ständig ihre auf den neuesten Stand gebrachten Daten bereithalten müssen, damit die übergeordneten Systeme jederzeit diese Daten lesen (abfragen) können.

Diese Methode verläuft völlig synchron, hat aber den Nachteil, daß sie periodisch wiederholt werden muß, damit Datenveränderungen rechtzeitig erkannt werden können.

Interrupt. Hierbei melden sich die *untergeordneten Systeme* selbständig über eine Interruptleitung, wenn sich ihre Daten entsprechend verändert haben. Das hat den Vorteil, daß das jeweils übergeordnete System seine Aufmerksamkeit nur dann auf die Informationserfassung lenken muß, wenn es neue Sachverhalte zu berücksichtigen gibt und es zum Datenempfang bereit ist. In der übrigen Zeit kann es eine andere Arbeit verrichten.

Die Übertragung der Stellsignale erfolgt beim hierarchischen System zwangsweise nach Ermessen des Masters. Die übergeordneten I-Systeme übergeben ihre Anweisungen an die ihnen untergeordneten I-Systeme, indem sie die Daten auf die Ausgabeleitungen legen und den entsprechenden Empfangsregistern den Übernahmeimpuls erteilen. Diese Form der Ausgabe wird als *synchrone Datenausgabe* bezeichnet.

Im hierarchischen System werden also die Zeitpunkte des Datenholens (Dateneingabe) und des Datensendens (Datenausgabe) weitestgehend vom übergeordneten I-System bestimmt. Ebenso bestimmt der Master die Art der Einwirkungen auf den zu steuernden Prozeß.

Anders sieht es beim *gleichberechtigten System* aus. Hier treten die Partner in einen zeittoleranten Informationsaustausch. Die Grundlage hierzu stellt das sog. Handshake-Verfahren dar.

Diejenige Baugruppe, die die Daten sendet, wird als Sender oder Talker (Sprecher) bezeichnet. Der Empfänger der Daten wird auch Listener (Hörer) genannt. Das Handshake-Verfahren vollzieht sich wie folgt:

- Der Sender kündigt dem Empfänger das Vorliegen von Daten an.
- Der Empfänger erklärt seine Bereitschaft zur Übernahme der Daten.
- Der Sender übergibt die Daten.
- Der Empfänger bestätigt die erfolgreiche Übernahme der Daten.

Wir erkennen, daß mit dieser Verfahrensweise die Möglichkeit gegeben ist, daß sowohl Sender als auch Empfänger zunächst noch die wichtigsten Arbeiten zu Ende führen können, ehe sie den angebotenen Datentransport realisieren. Natürlich ist die Nutzung dieser Möglichkeit eine Ermessensfrage und muß vom Programmierer im Sinne der Optimierung des Gesamtprozesses gehandhabt werden.

Wie kann nun die Signalisierung nach dem Handshake-Verfahren vollzogen werden?

Hierzu gibt es ebenfalls die zwei schon genannten Möglichkeiten. Entweder wird eine Benachrichtigungsleitung aktiviert, die der Empfänger von Zeit zu Zeit abfragt, ob schon eine Ankündigung vorliegt, oder man arbeitet mit Interrupt.

Zur Erläuterung der Interruptanwendung ist es das einfachste, wenn wir uns hierzu zwei Mikrorechner MR1 und MR2 vorgeben, die sich gegenseitig informieren sollen. Wir wissen, daß jeder Rechner sein bestimmtes Programm abzuarbeiten hat und daß er dabei am besten nicht gestört wird. Nun sollen aber beide Rechner auf die gegenseitige Information angewiesen sein, und wir setzen voraus, daß MR1 an MR2 eine Information senden will. Dann wird MR1 ein Interruptsignal an MR2 übertragen, welches von MR2 als Aufforderung verstanden wird, daß MR1 Daten übergeben will.

Ist MR2 bereit, die Daten von MR1 zu übernehmen, überträgt er seinerseits ein Interruptsignal an MR1. Dieses wird von MR1 dazu ausgewertet, nun die Daten an MR2 zu übergeben. Hat MR2 die Daten empfangen, so gibt er ein Quittungssignal. Damit ist dann die Übertragung beendet.

Je mehr Systeme informationstechnisch gekoppelt werden, um so dringender wird die Lösung der Frage nach der optimalen Verständigungsmethode zwischen den Systemen. Im folgenden Abschnitt wollen wir daher die Gestaltung des Unterbrechungssystems genauer betrachten.

3.10.8. Gestaltung des Unterbrechungssystems

Unter einer Unterbrechung versteht man den Vorgang, daß der Mikrorechner seine momentane Arbeit einstellt und auf die durch die Unterbrechungsanforderung ausgelöste Anweisung reagiert.

In der ersten Entwicklungsphase gab es für eine Unterbrechung nur den Begriff *Interrupt*.

Bei unserem einfachen Mikroprozessor war das Problem so gelöst worden, daß bei einem Impuls auf den Interrupteingang der Interrupt-RS-Trigger rückgesetzt wird und damit das Mikrosteuerwerk in den STOP-Zustand läuft. Aus diesem Zustand kann unser einfacher Mikroprozessor nur befreit werden, wenn man den Interrupt-RS-Trigger durch einen Impuls auf den Setzeingang wieder in den Zustand 1 bringt, ihn also neu startet. Diese Form der Interruptreaktion ist sehr primitiv und stellt nur ein Anhalten des Mikrosteuerwerks dar.

Für eine effektive Zusammenarbeit zwischen informationsverarbeitenden Systemen benötigt man jedoch eine Unterbrechungsmöglichkeit, die mehrere Reaktionsmöglichkeiten zuläßt. In **Tafel 3.24** sind in einer Übersicht die verschiedenen Interruptsignalformen, Anforderungen an ihre Übertragungswege und eine grobe Einschätzung der Anzahl der Reaktionsmöglichkeiten dargestellt.

Fall 1: ein Interruptimpuls, eine Interruptleitung. Hierbei läßt sich, wie im Beispiel des einfachen Mikroprozessors, nur eine Interruptreaktion realisieren. Anstelle der Einnahme des STOP-Zustands kann auch eine andere Reaktion erfolgen, wie z. B. die Abarbeitung eines Unterprogramms zur Ausführung bestimmter Arbeitsschritte.

In beiden Fällen muß das Mikrosteuerwerk des Rechners auf die Einleitung der Reaktion programmiert sein. So muß bei Einschaltung eines Interruptunterprogramms das Umschalten des Programmzählerstands auf die Adresse des Befehlsspeicherplatzes vorgenommen werden, auf dem der Anfangsbefehl des Reaktionsunterprogramms (interrupt-service-routine ≙ ISR) steht.

Tafel 3.24. Interruptsignale und Interruptreaktionen

Fall	Signalform	Anzahl der Interruptleitungen	Anzahl der Reaktionsmöglichkeiten
1	Impuls	eine	eine Reaktion; entweder STOP-Zustand einnehmen oder Umschalten auf ein Unterprogramm
2	Impuls	n	n verschiedene Reaktionen; mit jeder Leitung kann der Aufruf eines anderen Unterprogramms erfolgen
3	n-stelliges, binärcodiertes Signalwort	n bei Parallelübertragung 1 bei serieller Übertragung	2^n verschiedene Reaktionen, wenn für jede Kombination der Aufruf eines anderen Unterprogramms erfolgt
4	Sprachsignal	1 Leitung oder Schallwelle (serielle Übertragung)	sehr viele Reaktionsmöglichkeiten, wenn die Sprache ein entsprechend umfangreiches Vokabular und Rechnerprogramm entsprechender Vielfaltigkeit aufweist
5	Bildsignal	Lichtstrahl (Parallelübertragung)	sehr viele Reaktionsmöglichkeiten, wenn entsprechende Analyse- und Steuerprogramme vorliegen

Bild 3.67 demonstriert den Ablauf des beschriebenen Vorgangs. das Unterprogramm kann z. B. dazu dienen, daß der Rechner bestimmte Meßdaten holt, nach deren Auswertung er weitere Maßnahmen einleitet. Nach Erledigung dieser Arbeiten kann er in das Hauptprogramm zurückkehren oder sich selbst in den STOP-Zustand manövrieren.

Die Unterprogrammtechnik hat den großen Vorteil, daß der Programmierer die Reaktion nach den Erfordernissen gestalten kann. Wenn beispielsweise ein Kurzschluß eines Elektroenergie-Übertragungssystems die Ursache für die Interruptimpulsgabe war, kann der Rechner noch einige Male überprüfen, ob dieser gemeldete Kurzschluß nicht nur kurzzeitig zur Überlastung führt. Stellt sich z. B. heraus, daß er nur einige Millisekunden dauert, so brauchen nicht gleich rigorose Abschaltungsmaßnahmen eingeleitet zu werden.

Das Reaktionsunterprogramm muß so angelegt sein, daß im Betriebsfall keinerlei Daten des Hauptprogramms zerstört werden. Sollten z. B. Registerdaten durch das Unterprogramm gefähr-

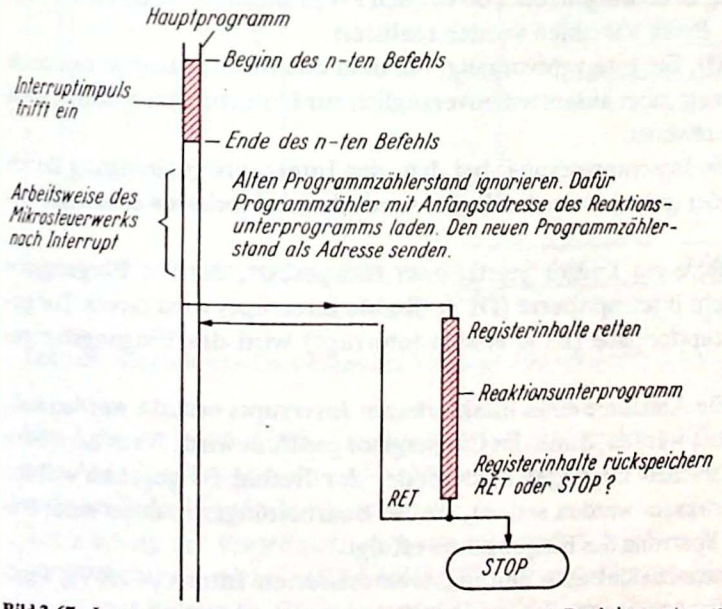


Bild 3.67. Interruptgestaltung bei einem Einzelimpuls durch Reaktionsunterprogramm

det sein, müssen diese zu Beginn weggespeichert und abschließend wieder zurückgeholt werden. Außerdem muß dafür gesorgt werden, daß während der Abarbeitung des Reaktionsunterprogramms der Interrupteingang gesperrt bleibt.

Fall 2: ein Interruptimpuls auf einer von *n* Leitungen. Es kann hierbei je Interruptleitung der gleiche Effekt wie beim Fall 1 erzielt werden.

Für jedes Signal ist dann im Prozessor eine bestimmte Adresse vorgesehen, ab der die Bearbeitung nach Annahme der Unterbrechung fortgesetzt wird **(Bild 3.68)**.

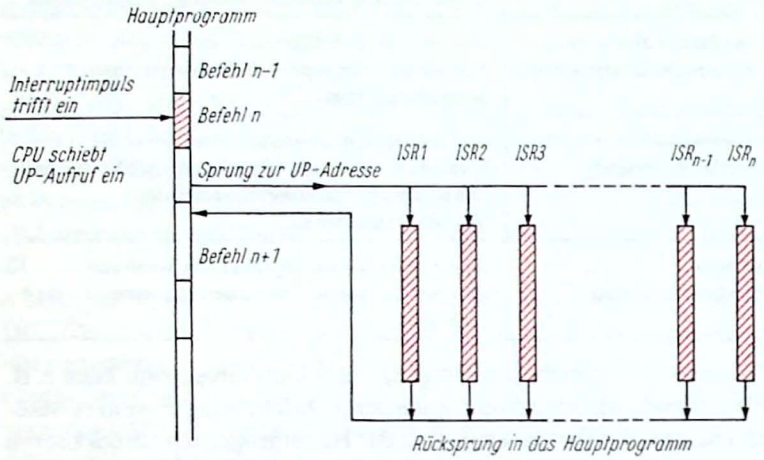


Bild 3.68. Interruptgestaltung bei n verschiedenen Impulsen durch Reaktionsunterprogramme (interrupt service routine ≙ ISR)

Da auch der Fall eintreten kann, daß mehrere Interruptquellen gleichzeitig eine Unterbrechung anfordern, muß eine Dringlichkeitsreihenfolge definiert werden. Jedem dieser Interrupteingänge muß eine bestimmmte Priorität (Rang) gegeben werden. Dann wird ein Interruptsignal auf dem Eingang mit der höchsten Priorität als erstes behandelt, danach erst die anderen, ebenfalls in der definierten Rangfolge.

Ein anderes Problem besteht darin, ob einem Interruptsignal gestattet werden sollte, sofort bei seinem Erscheinen die Unterbrechung zu erzwingen, oder ob es auch Programmteile geben sollte, die vor Unterbrechung geschützt sind. Beide Varianten wurden realisiert.

Nichtmaskierbarer Interrupt (NMI). Ein Interruptvorgang, bei dem das Interruptsignal nur noch der Prioritätseinschränkung unterliegt, aber ansonsten unverzüglich zur Unterbrechung führt, wird als nichtmaskierbarer Interrupt bezeichnet.

Maskierbarer Interrupt (INT). Ein Interruptvorgang, bei dem der Interruptsignaleingang durch Softwaresteuerung freigegeben oder gesperrt werden kann, wird als maskierbarer Interrupt bezeichnet.

Dabei wird durch besondere Befehle ein Trigger gesetzt oder rückgesetzt, der das Eingangstor freigibt oder sperrt. Mit dem Befehl Interruptsperre (DI ≙ disable interrupt) wird dieses Tor geschlossen, mit dem Befehl Interruptfreigabe (EI ≙ enable interrupt) wird das Eingangstor geöffnet.

Wenn also in einer Befehlsfolge die Annahme eines maskierbaren Interrupts erlaubt werden soll, muß als erstes der Befehl EI gegeben werden, damit das Eingangstor geöffnet wird. Auch nach oder während der Bearbeitung eines solchen Interrupts muß wieder der Befehl EI gegeben werden (wenn weitere Interruptsignale wirksam werden sollen), da bei Bearbeitungsannahme eines solchen Interrupts automatisch eine Sperrung des Eingangstors erfolgt.

Beim maskierbaren Interrupt unterscheidet man den nichtvektorisierten Interrupt (NVI) (entspricht dem Fall 1 oder 2) und den vektorisierten Interrupt (VI) (entspricht dem Fall 3).

Fall 3: n-stelliges binär codiertes Signalwort. Ein Unterbrechungssignal soll kurz sein, damit möglichst schnell reagiert werden kann. Es soll aber auch so aussagefähig sein, daß die optimalen Reaktionen zur Behebung der Unterbrechungsursache eingeleitet werden können. Je instruktiver die übermittelten Informationen sind, um so effektiver kann reagiert werden.
Sendet man nicht nur einen Impuls, sondern ein binär codiertes Wort, kann die Unterbrechungsursache wesentlich differenzierter übermittelt werden. Das erfordert aber entweder entsprechend viele Leitungen bei der Parallelübertragung oder den notwendigen Sende- und Empfangsaufwand bei einer seriellen Übertragung.
Vektorisierter Interrupt (VI). Um diese Problematik zu vereinfachen, hat man bisher das Verfahren des vektorisierten Interrupts realisiert. Dieser wird als maskierbarer Interrupt angemeldet.
Bei einfacheren Prozessoren stellt man diese Betriebsart durch Softwaresteuerung ein, moderne Prozessoren haben einen speziellen VI-Eingang.
Betrachten wir zur Erläuterung **Bild 3.69**. Nachdem die Anmeldung eines VI erfolgt ist ①, werden zunächst Steuersignale ② aktiv, die die den VI anmeldende Baugruppe veranlassen, ihrer VI-Adreßkomponente auf den Bus zu schalten, so daß die CPU diese im VI-Adreßregister einspeichern kann ③. Dann wird der bisherige Stand des Programmzählers in den STACK gerettet ④. Anschließend wird die VI-Adreßkomponente (als niederwertiger Adreßteil ≙ NWAT) zusammen mit der VI-Basisadresse (als höherwertiger Adreßteil ≙ HWAT) zum Interruptpointer ⑤ zusammengestellt. Damit steht dann eine Adresse zur Verfügung, mit der aus der ISR-Adreßtabelle im ROM die zugehörige ISR-Anfangsadresse geholt wird ⑥. Mit dieser wird dann der Programmzähler geladen ⑦, und die Abarbeitung der ISR (interrupt-service-routine) beginnt.

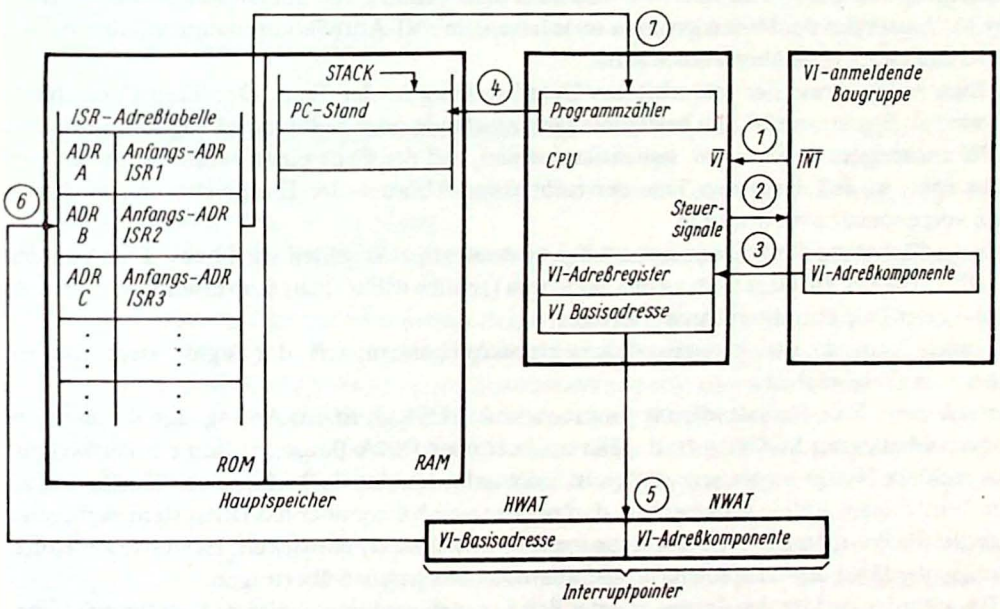


Bild 3.69. Schematische Darstellung der Vorgänge beim vektorisierten Interrupt

Das Arbeiten mit der ISR-Adreßtabelle bringt einen Vorteil für die Programmierung. Sollte einmal eine Adreßänderung notwendig sein, so brauchen nur die Angaben auf dem entsprechenden Platz in dieser Tabelle geändert zu werden.
Zur Klärung der Prioritätsfrage beim vektorisierten Interrupt wurde die Interrupt-Prioritätskette (daisy chain) entwickelt **(Bild 3.70)**. Die VI-anmeldenden Baugruppen sind spezielle Zusatzbausteine zum System (z. B. E/A- und CTC-Einheiten). Diese haben zum Aufbau der Daisy chain je

einen Interrupt-Prioritätsketten-Eingang IEI und einen -Ausgang IEO. Diejenige Baugruppe, die die höchste Priorität erhalten soll, bekommt an ihrem Eingang IEI ein festes Potential von +5 V (logisch 1). Der Eingang IEI der Baugruppe mit der nächsthöheren Priorität wird mit dem Ausgang IEO der ersten Baugruppe verbunden. So geht es mit den anderen Bausteinen weiter. Auf diese Weise entsteht die Prioritätskette, bei der man an der Lage in der Kette erkennen kann, welche Priorität eine Baugruppe hat.

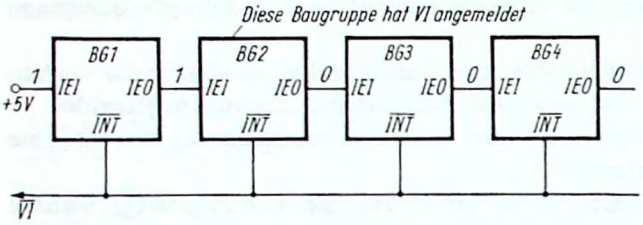


Bild 3.70
Schematische Darstellung der Interruptprioritätskette (daisy chain)

Wenn nun eine Baugruppe einen vektorisierten Interrupt anmeldet, wird das Potential an ihrem Ausgang IEO zu Null, wodurch auch alle weiter rechts liegenden Baugruppen an den Ein- und Ausgängen das Nullpotential erhalten. Damit verlieren sie ihre Berechtigung, eine Interruptanmeldung weiterzugeben.

Eine Baugruppe mit höherer Priorität ist dagegen noch in der Lage, einen Interrupt zu melden. Hat nun, wie im Bild 3.70 dargestellt, die Baugruppe BG2 einen VI durchgesetzt und ihre ISR ist in Bearbeitung, so kann eine Interruptanmeldung der Baugruppe BG1 die Bearbeitung dieser ISR unterbrechen, falls in der ISR der Baugruppe 2 der Befehl EI (enable interrupt) schon gegeben wurde. Der Übergang von IEI = 1 zu IEO = 0 wird auch dazu benutzt, die interruptanmeldende Baugruppe bei Aussenden der Steuersignale zu veranlassen, ihre VI-Adreßkomponente auf den Bus zu legen, so daß die CPU sie übernehmen kann.

Trap. Eine andere Form der vektorisierten Unterbrechung ist der Trap. Der Trap (Verschluß, Falle) wird als Signal benutzt, um bestimmte Grenzzustände oder bestimmte Programmierfehler der CPU anzuzeigen. So kann z. B. signalisiert werden, daß das Ende eines definierten Speicherbereichs naht, so daß durch den Trap der rechtzeitige Abbruch der Einspeicherung in diesen Bereich vorgenommen werden kann.

Der zur Auffächerung des Pointers notwendige niederwertige Adreßteil wird beim Trap von der CPU selbst geliefert. Im Gegensatz zu den asynchron (zeitlich willkürlich) eintreffenden Interruptsignalen ist der Trap ein taktsynchrones Ereignis.

Es gibt auch Traps, die von Zusatzbausteinen herrühren können, z. B. der Segmenttrap von der Speicherverwaltungseinheit.

Busanforderung. Eine Busanforderung (bus request ≙ BUSRQ) ist ein Antragssignal von informationsverarbeitenden Systemen (z. B. Mikrorechner oder DMA-Baustein), um ein Bussystem, das für mehrere Nutzer zugelassen worden ist, dem anfordernden Teilnehmer zur Verfügung zu stellen. Wird diesem Antrag stattgegeben, darf nur der eine Nutzer über das Bussystem verfügen. Um hierbei die Prioritäten zu regeln, wurden spezielle *Busverwalter* entwickelt. Es wurde aber auch das Prinzip der Daisy chain auf solche busanfordernden Baugruppen übertragen.

Eine Busanforderung läßt sich deswegen unter Fall 3 (n-stelliges binär codiertes Signalwort) systematisieren, weil nach der erteilten Bestätigung zur Busbenutzung (bus acknowledge ≙ BUSAK) eine Datenübermittlung von mindestens einem Wort, oftmals aber auch eine Übertragung ganzer Datenblöcke erfolgt.

Die von der Busbenutzung in dieser Zeit ausgeschlossenen Systeme müssen ihre an das Bussystem angeschlossenen Tristate-Tore dann alle in den hochohmigen Zustand schalten bzw. in diesem belassen. Bei Bestätigung einer Busanforderung werden also bis auf den Busbenutzer alle anderen Systeme in ihrer Arbeitsmöglichkeit eingeschränkt. Wenn es nur einen Bus gibt, bedeutet dies, daß alle anderen Systeme nicht miteinander in Kommunikation treten können.

Das hat zu der Überlegung geführt, das Ein-Bussystem bei Mehrrechnerkopplung zu verlassen. Der Ausweg besteht darin, die Rechner mit mehreren E/A-Kanälen zu versehen und jeweils mehrfach zu verkoppeln. Mit vier E/A-Kanälen je Rechner lassen sich dann Matrixstrukturen **(Bild 3.71)** realisieren. Mikrorechner, die speziell für solche Mehrfach-Kommunikationsmöglichkeiten vorbereitet sind, werden als *Transputer* bezeichnet.

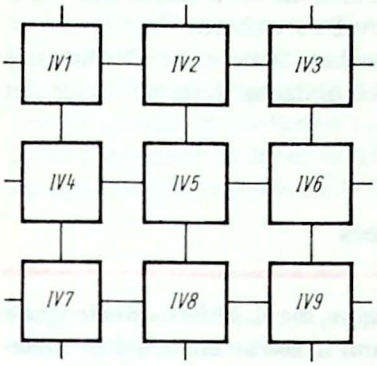


Bild 3.71
Prinzip der Mehrfachkommunikation zwischen Rechnern in Matrixstruktur

Bei all den verschiedenen Unterbrechungsanforderungen besteht die Frage, wie man die Priorität der einzelnen Unterbrechungsarten untereinander gelöst hat. Es hat sich folgende Rangfolge entwickelt:

1. Busanforderung (wird nach jedem Maschinenzyklus auf Vorliegen einer Anmeldung abgefragt)
2. Trap
3. nichtmaskierbarer Interrupt
4. Segmenttrap
5. vektorisierter Interrupt
6. nichtvektorisierter Interrupt

} sind maskierbar

Die weitere Entwicklung bei der Gestaltung der Unterbrechungssysteme wird sicherlich noch recht interessante Varianten hervorbringen.
Dazu gehören auch die Möglichkeiten, die aus den Fällen 4 und 5 der Tafel 3.24 hervorgehen werden.

Fall 4: Sprachsignal. Die Sprachsignalübertragung bei Interrupt ähnelt der Übertragung nach Fall 3, wenn man dort die serielle Übertragung anwendet. Das Sprachsignal ist jedoch wesentlich komplizierter als ein binär codiertes Signalwort stukturiert und erfordert daher einen höheren Aufwand bei seiner Herstellung (Synthese) und seiner Auswertung (Analyse). Der große Vorteil liegt darin begründet, daß der Mensch unmittelbar mit seiner Sprache dem Rechner Befehle erteilen, vom Rechner Erläuterungen bekommen und auch die Kommunikation zwischen den Rechnern abhören kann.
Die Verwendung des Interruptprinzips zur Übertragung von Sprachsignalen zeigt deutlich den Wandel im Gebrauch des Interrupts vom ursprünglichen *Ausschalten* bei Alarmzuständen bis zur interruptgesteuerten Unterhaltung.
Bild 3.72 zeigt ein grobes Blockbild zu diesem Verfahren.

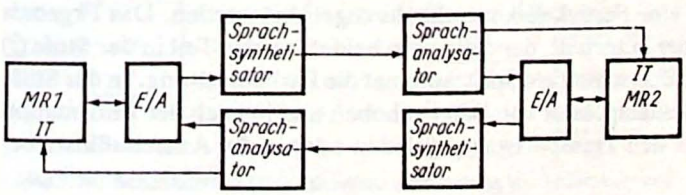


Bild 3.72. Grobes Blockbild zur Sprachkommunikation zwischen Mikrorechnern

Fall 5: Bildsignal. Ein Bildsignal kann einen hohen Informationsgehalt aufweisen. Zur effektiven Auswertung dieser Informationen muß aber ein dem menschlichen Bildauswertungssystem ähnliches Verfahren herangezogen werden. Dazu gehört ein Raster aus etwa 10^8 optischen Sensorelementen und ein Informationsverarbeitungssystem, das in der Lage ist, die wesentlichen Bildinformationen auszuwählen.
Das System muß also Abstraktionsvermögen besitzen. Dann können die Ergebnisse einer optischen Prozeßüberwachung die Analyse des Gesamtprozesses wertvoll bereichern.
Obwohl es noch großer Anstrengungen bedarf, um mit technischen Mitteln die Qualität des menschlichen Bilderkennungssystems zu erreichen, können doch einfache Systeme dieser Art schon heute mit Erfolg eingesetzt werden.

3.11. Anwendungsmöglichkeiten des Mikrorechners

Wir wissen: Der Mikrorechner ist eine mikroelektronische Baugruppe, die elektrische Binärsignale aufnimmt und nach einem vorgegebenen Algorithmus (Programm) in solche elektrischen Binärsignale umformt, die zur Steuerung eines technischen Prozesses dienlich sind.
Da viele Prozesse analoge Meß- und Stellsignale erfordern, werden oftmals Analog-Digital-Umsetzer (ADU) vorgeschaltet und Digital-Analog-Umsetzer (DAU) nachgeschaltet.
Zur außerplanmäßigen Information von Mikrorechnern werden Interruptanforderungen gesendet, die durch besondere Unterprogramme bearbeitet werden.

Die Vorteile des Mikrorechners bestehen darin, daß er

- ein außerordentlich kleines Volumen hat
- nur geringe Energie benötigt
- durch Programmierung an viele Einsatzprobleme anpassungsfähig ist
- relativ billig ist und daher kostengünstig Mehrrechnersysteme aufgebaut werden können.

Sein Nachteil gegenüber speziellen Schaltkreisen zur Informationsverarbeitung besteht darin, daß die vom Mikrorechner bei der Programmabarbeitung zu durchlaufenden Arbeitszustände eine bestimmte Zeit erfordern.
Seine Programmabarbeitungsgeschwindigkeit ist aber für viele Einsatzfälle hinreichend groß. Außerdem werden durch Weiterentwicklung der technologischen Prozesse noch schnellere Typen entstehen.
Seine Hauptanwendung liegt auf dem Gebiet der Steuerung technischer Prozesse in der Industrie. Er kann dort eingesetzt werden bei der Steuerung von Bearbeitungsautomaten aller Art, besonders auch von Schweißrobotern und Manipulatoren. Es können aber auch Beschickung und Entladung über Fließbänder oder automatisch gesteuerte Transportfahrzeuge durch Mikrorechner dirigiert werden. **Bild 3.73** zeigt eine Skizze für eine solche Anwendung.
In der Bearbeitungsstufe ① verbindet ein Schweißroboter die Teile. Der Mikrorechner des Roboters steuert die Bewegung der Schweißbrenner und optimiert die den Schweißprozeß beeinflussenden Parameter. Anschließend kann eine Festigkeitskontrolle durchgeführt werden. Das Ergebnis dieser Kontrolle wird dem Leitrechner mitgeteilt, der dann entscheidet, ob das Teil in der Stufe ② noch bearbeitet wird. In dieser Stufe übernimmt ein Spritzautomat die Farbgestaltung. In der Stufe ③ wird das Teil durch einen Umlademanipulator von Band gehoben und je nach der Information durch den Leitrechner entweder auf den Transportwagen geladen oder in die Ausschußkiste befördert.
Dieser Prozeß läßt sich auch so einrichten, daß innerhalb der Ausschußteile noch Sortierungen vor-

genommen werden in Teile, die nachbearbeitbar sind, und in solche, die auf den Schrotthaufen wandern.

Auch Prozesse der chemischen Verfahrenstechnik lassen sich durch Mikrorechner steuern. Im **Bild 3.74** ist ein solcher Prozeß in allgemeiner Art skizziert, der einen groben Einblick in die gegebenen Möglichkeiten verschaffen soll.

Es sollen die drei Stoffkomponenten A, B, C dosiert und im Mischer vorgewärmt und homogen verteilt werden. Dosierung und Temperaturanstieg werden vom Mikrorechner überwacht. Nach einer vorgegebenen Zeit wird das Gemisch in den Reaktor befördert, in dem die volle Temperatur und der nötige Druck erzeugt werden. Dann wird die Komponente D hinzugegeben und das ganze Gemisch bestimmten zeitlichen Temperaturverläufen ausgesetzt. Anschließend kommt das Produkt in die Trennstufe, in der es mit Hilfe des Stoffes H in die zwei Komponenten E1 und E2 zerlegt wird, die das Fertigprodukt darstellen.

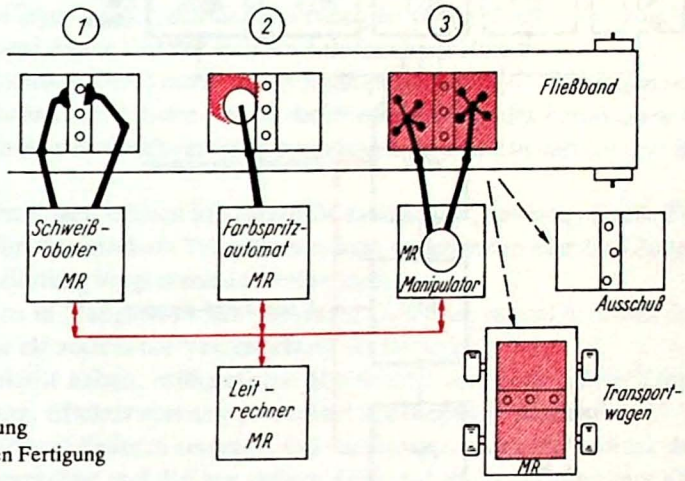


Bild 3.73
Schematische Darstellung zur Anwendung von Mikrorechnern in der automatischen Fertigung

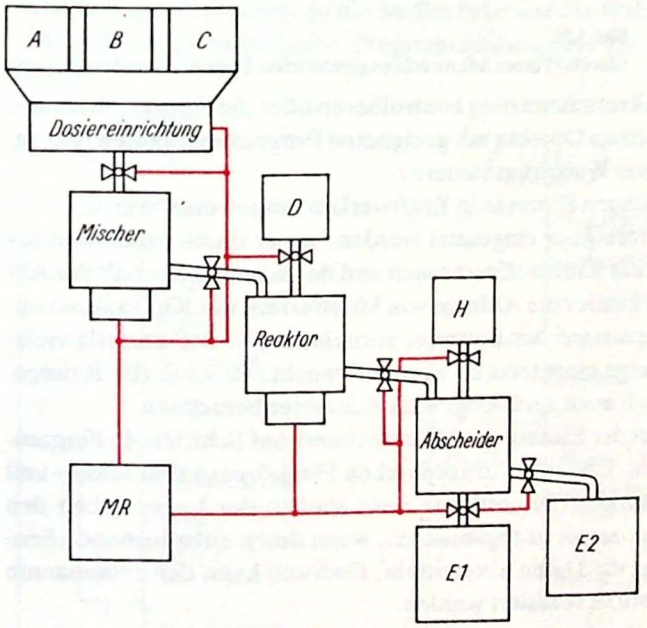


Bild 3.74. Schematische Darstellung zur Anwendung von Mikrorechnern in automatischen Anlagen der chemischen Industrie

Bei diesem Prozeß übernimmt der Leitrechner die Kontrolle aller Teilschritte und gibt den einzelnen Stellgliedern Optimierungshinweise. Bei Prozessen, bei denen durch Ausfall der Steuerung oder durch Fehlsteuerungen große Werte verlorengehen können, ist es ratsam, den Prozeß mehrfach zu kontrollieren und noch Prozeßwächter und Einrichtungen der konventionellen Steuerungstechnik wirken zu lassen, damit im Fall des Versagens des Mikrorechners nicht der Prozeß gefährdet wird.

Besonders wirksam kann der Mikrorechner zur Überwachung von Gefahrenherden eingesetzt werden, indem er laufend Meßergebnisse einholt und den Trend eines Prozesses überwacht **(Bild 3.75)**. So kann er bei Überwachung von Patienten auf Intensivstationen bei Gefahr nicht nur Alarm, sondern sofort auch durch Zusammenstellen der kritischen Daten einen Situationsbericht geben, so daß der Arzt schneller diagnostizieren kann.

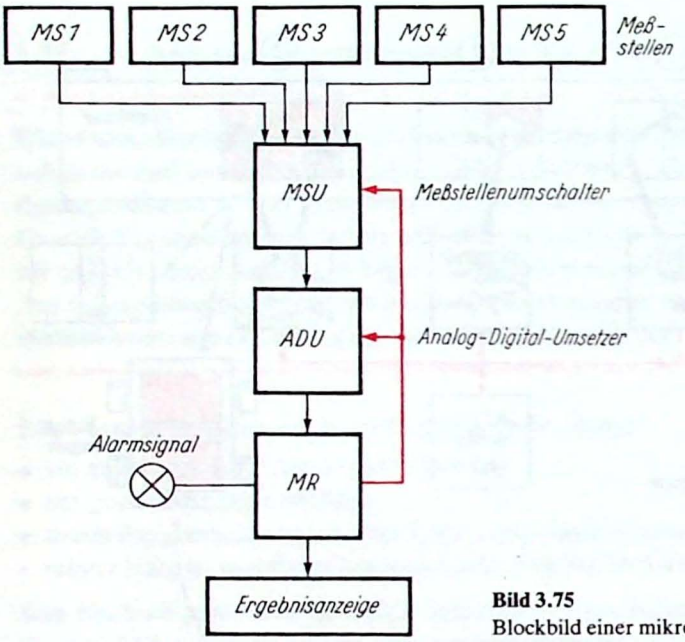


Bild 3.75
Blockbild einer mikrorechnergesteuerten Überwachungseinrichtung

So können Mikrorechner auch die Umweltverschmutzung kontrollieren oder die Klimaverhältnisse in Treibhäusern beobachten und die einzelnen Objekte mit geeigneten Programmabläufen (Beregnung, UV-Bestrahlung, Düngung) in ihrem Wachstum steuern.

Ebenso können Komplexdaten der vielfältigen Prozesse in Kraftwerken ausgewertet werden.

Auch in Kraftfahrzeugen kann der Mikrorechner eingesetzt werden, wo er durch Steuerung des Zündeinsatzes dazu beitragen kann, daß der Kraftstoffverbrauch und der Schadstoffgehalt der Abgase gesenkt werden können. Außerdem kann er die Abfrage von Meßwerten, wie Kühlwassertemperatur, Benzin- und Ölstand oder Ladezustand der Batterie, vornehmen, so daß anstelle vieler Instrumente nur eine Leuchtsymbolanzeige eingesetzt zu werden braucht. Er kann die Reisegeschwindigkeit oder die mit dem Tankinhalt noch zurücklegbaren Kilometer berechnen.

Noch wichtiger als beim Kraftfahrzeug ist der Einsatz des Mikrorechners auf Schiffen, in Flugzeugen und vor allem in Weltraumfahrzeugen. Besonders in kosmischen Flugkörpern sind Masse- und Energieeinsparungen von größter Bedeutung. Die Mikrorechner sind in der Lage, neben den Steuerungs- sogar auch die Erkundungsprozesse zu organisieren, wenn ihnen entsprechende Sensoren (Meß- und Wahrnehmungsglieder) die Daten übermitteln. Dadurch kann der unbemannte Weltraumflug zur Erkundung ferner Gestirne realisiert werden.

Dieses Beispiel zeigt, daß man Mikrorechner einsetzen kann, um den Menschen vor gesundheitlichen Schädigungen oder unzumutbaren Bedingungen zu bewahren. So können mikrorechner-

gesteuerte Systeme Beobachtungsergebnisse aus großen Meerestiefen, tiefen Bohrlöchern, Höhlenlabyrinthen u. a. ermitteln. Dabei sind allerdings besondere Maßnahmen zu treffen, damit sämtliche eingesetzten Apparaturen die dabei auftretenden extremen Bedingungen, wie hohe Temperaturen, hohe Drücke, Feuchtigkeit usw., vertragen. Gegebenenfalls muß ein System von Schutzmaßnahmen, wie Kühlung, Panzerung, Hermetisierung, zur Anwendung kommen.
Ein großes Einsatzgebiet wird dem Mikrorechner in der Nachrichten- und Meßtechnik erschlossen.
Nehmen wir zunächst die Fernsprechvermittlungstechnik. Hier tritt das Problem auf, daß in Spitzenbelastungszeiten oder bei Abwesenheit des Teilnehmers Anrufversuche scheitern und der Anrufer etliche Zeit verwenden muß, um die Verbindung herzustellen. Hier kann der Mikrorechner helfen. Man kann ihm die volle Arbeit beim Anruf übertragen. Natürlich muß man ihm vorher die Rufnummer des gewünschten Teilnehmers eingeben. Alles weitere wird dann automatisch erledigt. Der Anrufer wartet nur, bis das Signal ertönt, das die Herstellung der gewünschten Verbindung bekanntgibt, kann sich also während dieser Zeit mit anderen Arbeiten beschäftigen.
Damit nicht durch zu langes automatisches Aussenden der Rufimpulse Leitungsblockierungen verursacht werden, müssen Einschränkungen in der Anzahl der Wiederholungen des Anrufs gemacht werden. Oder man speichert die Anrufwünsche in den Vermittlungsämtern und steuert von dort die Vermittlungsversuche.
Bei mobilen UKW-Funksprechanlagen können Mikrorechner dafür sorgen, daß eine variable Zuteilung von Sendefrequenzen für die einzelnen Teilnehmer erfolgt, so daß in den einzelnen Zonen eine Optimierung der Kanalauslastung vorgenommen werden kann.
Der Einsatz von Mikrorechnern in Meßgeräten führt zu neuen Qualitäten sowohl bezüglich der Auswertung der Meßergebnisse als auch in der Vereinfachung der Bedienung.
So können, wie wir kennengelernt haben, arithmetische Mittelwerte, aber auch andere Kenngrößen, wie Standardabweichung, Effektivwert usw., berechnet und dargestellt werden.
Die Bedienungserleichterung kommt dadurch zustande, daß der Benutzer durch Tastendruck das ausgewählte Meßprogramm einschaltet und daß von diesem Zeitpunkt ab der Mikrorechner alle Abgleich- und Einstellvorgänge automatisch durchführt. Dem Bediener obliegt dann nur noch die Anschaltung des Geräts an die Meßstrecke und die Wahl der Programmtaste.
Bild 3.76 zeigt zwei grobe Programmablaufpläne für ein mikrorechnergesteuertes Digitalvoltmeter.

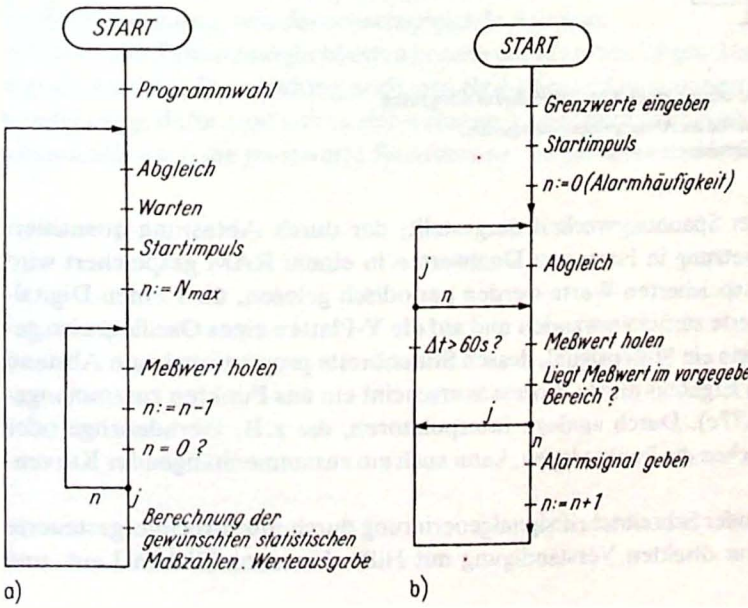


Bild 3.76
Grobe Programmablaufpläne für mikrorechnergesteuerte Digitalvoltmeter
a) zur Berechnung statistischer Maßzahlen
b) zur Grenzwertüberwachung

Bei Funktionsgeneratoren führt der Einsatz der Mikrorechner zu einer völlig neuen Qualität. So kann man durch ihn die Erzeugung beliebiger Funktionsformen steuern. Man braucht sich also nicht mehr auf Sinus-, Rechteck-, Dreieck- oder sonstige Elementarsignale zu beschränken, sondern kann beliebige Signale generieren. Das geht bis zu musikalischen Klangfolgen, Schreibschrift- und Sprachsignalen. Als Beispiel für einen Funktionsgenerator haben wir den Mustergenerator (S. 135) kennengelernt, der zwar mit festverdrahteter Logik dargestellt worden ist, aber ohne weiteres auch mit einem Mikroprozessor realisiert werden kann.
Diese Entwicklungsrichtung führt auch auf eine neue Art von Oszillografen, die sich vor allem zur Oszillogrammspeicherung eignen **(Bild 3.77)**.

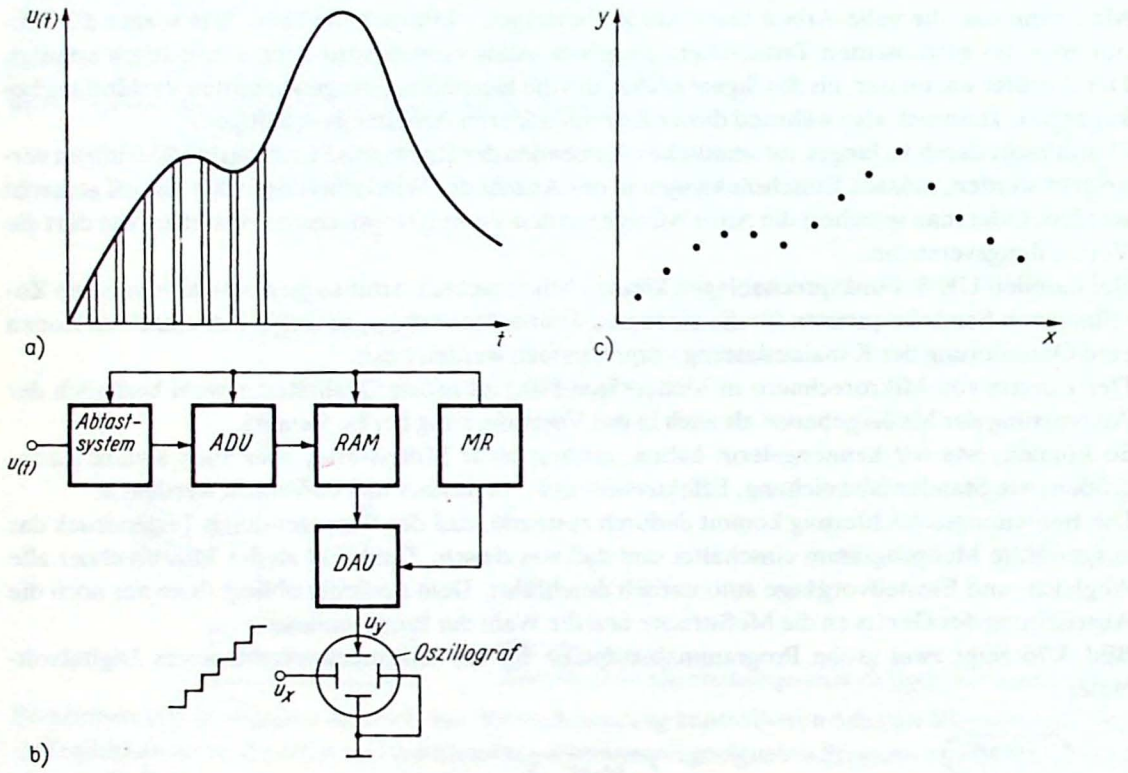


Bild 3.77
Anwendung des Mikrorechners zur Steuerung eines Abtastspeicheroszillografen
a) abzutastender Spannungsverlauf; b) Blockbild des Abtastspeicheroszillografen;
c) auf dem Bildschirm erscheinendes Oszillogramm

Im Bild 3.77a ist ein zeitlicher Spannungsverlauf dargestellt, der durch Abtastung quantisiert und nach Analog-Digital-Umsetzung in Form von Dualwerten in einem RAM gespeichert wird (Bild 3.77b). Die im RAM gespeicherten Werte werden periodisch gelesen, über einen Digital-Analog-Umsetzer in Analogwerte zurückverwandelt und auf die Y-Platten eines Oszillografen gegeben. Die X-Platten bekommen ein Stufensignal, dessen Stufenbreite proportional zum Abstand der Abtastzeitabstände ist. Im Ergebnis dieses Prozesses erscheint ein aus Punkten zusammengesetztes Oszillogramm (Bild 3.77c). Durch analoge Interpolatoren, die z. B. Geradenzüge oder andere Näherungskurven zwischen die Punkte legen, kann auch ein zusammenhängender Kurvenverlauf erzeugt werden.
Die bereits erwähnte Sprach- oder Schreibschriftsignalgenerierung durch mikrorechnergesteuerte Systeme führt dazu, daß es zur direkten Verständigung mit Hilfe der menschlichen Laut- und

Schriftsprache zwischen Mensch und Rechner kommt. Um diese Verständigung voll zu perfektionieren, bedarf es allerdings größerer Speicherkomplexe, so daß wir hier Zeuge einer allmählichen Vervollkommnung werden. Insgesamt werden hierzu Sprach- und Schriftgeneratoren sowie Sprach- und Schrifterkennungssysteme benötigt. Bei den Erkennungssystemen geht es ähnlich wie beim digitalen Abtastoszillografen zunächst um eine Zergliederung des Signals, wobei hier aber Verfahren eingesetzt werden, die summarische Eindrücke sammeln und auswerten und nach Möglichkeit nur soweit wie nötig ins Detail gehen.

Im Zusammenhang mit Sprach- und Schrifterkennungssystemen soll der übergeordnete Komplex der Diagnoseverfahren erwähnt werden. In verallgemeinerter Aussage geht es dabei darum, daß der Mikrorechner aus einer Fülle von Fakten logische Schlußfolgerungen zieht.

Solche Diagnoseverfahren können überall dort eingesetzt werden, wo es um die Bewertung von qualitativen und quantitativen Zuständen geht. Anwendungsmöglichkeiten sind gegeben bei

- Diagnose von Krankheiten
- Diagnose von fehlerhaften Zuständen in technischen Systemen (z. B. Produktionsanlagen, Kraftfahrzeuge, elektronische Geräte)
- Diagnose von Wetterverläufen
- Bewertung der Qualität von Erzeugnissen.

Auch diese Diagnosesysteme benötigen umfangreiche Speicher, in denen Algorithmen und Erfahrungswerte abgelegt sind.

Diagnosesysteme bilden die Grundlage für die automatische Fehlerbehebung, die sich in immer stärkerem Maße durchsetzen wird.

Auch der Bereich der Haushalts- und Heimgeräte wird vom Mikrorechnereinsatz erfaßt. So können Küchenmaschinen, Küchenherde, Backöfen, Nähmaschinen von Mikrorechnern programmgesteuert werden, wobei die Hausfrau nicht etwa in Assemblersprache programmieren können muß, sondern einfach Schalter und Tasten bedient.

Fernseh- und Phonogeräte erreichen ebenfalls mit Mikrorechnern einen hohen Bedienungskomfort: Sendereinstellung nach Programmzeitankündigung, automatische Farb- und Kontrastkorrektur, automatische Auswahl einzelner Abschnitte von Platten- und Tonbändern (d. h. eigene Programmzusammenstellungen) und automatische Übernahme von Sendungen durch Aufzeichnungsgeräte.

Nicht zu vergessen sind die Fernsehspiele, deren Vielfalt noch wesentlich wachsen wird, oder andere Spielautomaten, wie der schachspielende Automat.

Die Palette der Einsatzmöglichkeiten konnte nur in groben Zügen skizziert werden. Sicherlich wird durch die künftige Entwicklung noch eine Reihe neuer Anwendungen erschlossen.

Voraussetzung dafür sind neben der weiteren Vervollkommnung der Mikrorechner und der Speicherkapazität auch die preiswerte Realisierung von Sensoren und Stellgliedern.

4. Erläuterung spezieller Begriffe

Akkumulator. Register, das die Ergebnisse von arithmetischen und logischen Operationen speichert

Analog-Digital-Umsetzer (ADU). System, das Momentanwerte von analogen Signalen abtastet und in digital codierter Form darstellt

Arithmetik-Logik-Einheit (ALU). Baugruppe für die arithmetische und logische Verknüpfung von Variablen

Assemblersprache. Einfache synthetische Sprache, die die Programmierung von Mikrorechnern erleichtert (assembler ≙ Monteur)

Assemblierung. Rechnergesteuerte Übersetzung eines in Assemblersprache geschriebenen Programms in die Maschinensprache des verwendeten Mikrorechners

Bit. Abk. vom engl. binary digit; Bezeichnung für eine einzelne Stelle eines Binärworts; besonders bezeichnet werden das MSB (Abk. von engl. most significant bit), das höchstwertige Bit, und das LSB (Abk. von engl. least significant bit), das niederstwertige Bit

Beispiele: eine Datenwortbreite von 8 Bit, eine Speicherkapazität von 2048 Bit = 2 KBit, (hierbei entspricht der Buchstabe K der Zahl 1024)

Bitmuster. Muster, das durch die jeweilige Stellenbelegung (0 oder 1) in einem Binärwort entsteht

Bus. Sammelschiene; Anordnung von mehreren parallelen Leitungen zur gleichzeitigen Übertragung von Binärinformationen, dadurch gekennzeichnet, daß alle zum Informationsaustausch vorgesehenen Sender und Empfänger über Torglieder angeschlossen sind

Byte. Bezeichnung für ein bestimmtes Binärwort von 8 Bit Breite

Beispiel: eine Speicherkapazität von 1024 Byte = 1 KByte

Codierung. Verschlüsselung; Vorgang zur Umwandlung einer Information in ein Signal

CPU. Central processing unit; zentrale Verarbeitungseinheit eines Mikrorechners, gleichbedeutend mit dem Begriff Mikroprozessor

Decodierung. Entschlüsselung; Rücktransformation von codierten Zeichenfolgen in das Ursprungszeichensystem

Digital-Analog-Umsetzer (DAU). System, das digital codierte Werte in entsprechende Beträge eines analogen Signals umsetzt

EPROM. Abk. von engl. erasable programmable ROM, löschbarer, programmierbarer ROM

Ergibtzeichen := Abart des Gleichheitszeichens, mit dem angedeutet wird, daß die Gleichheit erst hergestellt werden soll; ⟨A⟩ := ⟨B⟩ bedeutet, daß der Inhalt des Speichers A aus dem Inhalt des Speichers B gebildet werden soll

Handshake-Verfahren. Verfahren zum Datenaustausch zwischen zwei Systemen, bei dem mit Hilfe vereinbarter Signale die Ausführung aller Teilschritte wechselweise angefordert und quittiert wird

Holografischer Speicher. System zur Speicherung räumlicher Bilder in einer Fotoplatte. Bei der Aufnahme werden durch die Überlagerung eines vom Objekt beeinflußten Strahls und eines Referenzstrahls ein und derselben kohärenten Lichtquelle Linienmuster (Überlagerungsmuster) in der Fotoschicht erzeugt; bei der Wiedergabe wird die Speicherschicht von dem gleichen kohärenten Licht durchstrahlt; dabei wird der Lichtstrahl genauso moduliert, wie es der Strahl war, der die Aufnahme bewerkstelligte; dadurch kann das von diesem Strahl getroffene Auge alle Einzelheiten nachempfinden; der holografische Speicher läßt sich auch zur Speicherung und Wiedergabe von Binärzeichen verwenden

Information. Auskunft, Benachrichtigung, Nachricht, Mitteilung, Belehrung, auswertbarer Inhalt von Neuigkeiten, aber auch der Vorgang der Übermittlung von Neuigkeiten

Informationsübertragung, parallel. Gleichzeitige Übertragung von Informationen auf einer Vielzahl von Übertragungskanälen; genaugenommen gibt es keine reine Parallelübertragung, sondern nur die zeitliche Aufeinanderfolge paralleler Informationsschübe

Informationsübertragung, seriell. Zeitlich nacheinander folgende Übertragung der einzelnen Teile einer Nachricht (Information)

Informationsverarbeitung. Prozesse, bei denen Informationen gesammelt, gespeichert, ausgewählt, verdichtet und mit Hilfe spezieller Verfahren ausgewertet werden

Interrupt. Unterbrechung; ein Interruptsignal dient zur Anforderung der Unterbrechung eines in Abarbeitung befindlichen Programms, damit Informationen außerhalb der geplanten Abarbeitungsreihenfolge zur Wirkung kommen

LSI. Abk. von engl. large scale integration; Großintegration von Bauelementen (10^3 bis 10^4 Elemente auf einem Chip)

Mikroprozessor. LSI-Schaltkreis, der die Funktion der zentralen Verarbeitungseinheit eines Prozeßrechners darstellt

Mikrorechner. Prozeßrechner, der mit integrierten Schaltkreisen auf wenigen Leiterkarten oder nur einer Leiterkarte oder sogar nur auf einem Chip realisiert ist

Mnemonik. Die Unterstützung der Gedächtnisleistung durch merktechnische Hilfsmittel, z. B. besondere Wörter oder Merksätze (Eselsbrücken)

MSB. Abk. von engl. most significant bit, höchstwertiges Bit eines Wortes

MSI. Abk. von engl. medium scale integration; Mittelintegration von Bauelementen (10^2 bis 10^3 Elemente auf einem Chip)

PLA. Abk. von engl. programmable logic array, programmierbare Logikanordnung; nach Wunsch des Anwenders programmierbarer logischer Zuordner

PROM. Abk. von engl. programmable read only memory, programmierbarer ROM

RAM. Abk. von engl. random access memory, Schreib-Lese-Speicher

Register. Kleine Speicheranordnung für eine nicht allzu große Anzahl von Bitstellen, meist für ein Binärwort

Befehlsregister. Register, das den jeweils aktuellen zu entschlüsselnden Befehl speichert

Flagregister. Register, das das Eintreffen von besonderen Zuständen signalisiert; die Flags (einzelne Trigger in dem Flagregister) werden zur Realisierung bedingter Programmverzweigungen verwendet

ROM. Abk. von engl. read only memory, Nur-Lese-Speicher, Festwertspeicher; eigentlich ein Zuordner, der auf jede mögliche Eingangsadresse ein bestimmtes Bitmuster liefert

Schieberegister. Register, bei dem die Information in den einzelnen Speicherelementen mit dem Schiebetakt um jeweils eine Stelle weitergeschoben wird

Signal. Gemusterter, materieller Träger, der zur Informationsübertragung verwendet wird

Spike. Kurzer, meist unerwünschter Zusatzimpuls

Sprung. Als Sprung bezeichnet man die Fortsetzung einer Programmabarbeitung bei einer durch den Sprungbefehl angegebenen Adresse; bei einem bedingten Sprung wird durch eine Zusatzbedingung (Wertebelegung eines Flags) vom Automaten entschieden, ob der Sprung ausgeführt wird; ein unbedingter Sprung wird bedingungslos ausgeführt

SSI. Abk. von engl. small scale integration; Kleinintegration von Bauelementen (10 bis 100 Elemente auf einem Chip)

STACK. Stapel; Speicher, der zur zwischenzeitlichen Aufbewahrung von Datenblöcken dient, die ähnlich wie bei einem Warenstapel eingespeichert und abgebaut werden; Prinzip: last in, first out; was zuletzt eingespeichert wurde, geht als erstes wieder heraus

Synchron. Gleichzeitig, im Rhythmus eines Taktes ablaufend

Trigger. Baugruppe, die bei Vorliegen bestimmter Pegel einen von zwei Zuständen einnehmen kann

Tristate-Gatter. Gatterbaustein mit den drei Ausgangszustandsmöglichkeiten 1, 0, hochohmige Sperrung des Ausgangs

Übertrag. (engl. carry), Wert, der bei arithmetischen Verknüpfungen von Ziffern in die nächsthöhere Ziffernstelle übertragen wird; der Übertrag aus der definierten Zahlwortbreite hinaus wird ebenfalls als Carry bezeichnet

Umcodierung. Transformation der Informationsdarstellung von einem Zeichensystem in ein anderes

VLSI. Abk. von engl. very large scale integration; sehr großer Integrationsgrad von Bauelementen (mehr als 10^4 Elemente auf einem Chip)

Wirtsrechner. Digitalrechner, der die Übersetzung eines in einer Programmiersprache geschriebenen Programms vornimmt, das nicht für seinen Maschinentyp bestimmt ist

Zeichen. Element einer Menge, die zur Übertragung von Nachrichten verwendet wird

Zuordner. Übergeordneter Begriff für Codierer, Umcodierer, Decodierer

Zustand. Gesamtheit der erkennbaren Eigenschaften eines materiellen Gebildes oder eines materiellen Prozesses zu einem bestimmten Zeitpunkt

Zweierkomplement. Besondere Darstellungsform einer Dualzahl, um die Subtraktion mit einem Addierwerk ausführen zu können; ist z die gegebene Dualzahl mit

$$z = \sum_0^n a_i B^i,$$

so wird das Zweierkomplement z_{2k} gebildet durch

$$z_{2k} = B^{n+1} - z,$$

wobei

$$B^{n+1} = 1 \cdot B^n + 1 \cdot B^{n-1} + \ldots + 1 \cdot B^0 + 1$$

ist.

Somit wird

$$\begin{aligned} z_{2k} &= B^{n+1} - \sum_0^n a_I B^i \\ &= \sum_0^n 1B^i + 1 - \sum_0^n a_i B^i \\ &= \sum_0^n (1 - a_i)\, B^i + 1 \\ &= \sum_0^n \bar{a}_i B^i + 1. \end{aligned}$$

Literaturverzeichnis

Eckhardt, D.; Konrad, E.; Leupold, W.: Digitale Grundschaltungen und ihre Anwendung. 2. Aufl. REIHE AUTOMATISIERUNGSTECHNIK, Bd. 161. Berlin: VEB Verlag Technik 1977

Eckhardt, D.; Konrad, E.; Leupold, W.: Entwurf komplexer digitaler Schaltungen. REIHE AUTOMATISIERUNGSTECHNIK, Bd. 175. Berlin: VEB Verlag Technik 1976

Stürz, H.; Cimander, W.: Logischer Entwurf digitaler Schaltungen. Berlin: VEB Verlag Technik; Heidelberg: Dr. Alfred Hüthig Verlag 1976

Kühn, E.: Handbuch TTL- und CMOS-Schaltkreise. Berlin: VEB Verlag Technik 1985

Möschwitzer, A.: Grundkurs Mikroelektronik, Standardschaltkreise, Kundenschaltkreise. Berlin: VEB Verlag Technik 1984

Entreß, G.; Entreß, L.: Einführung in die Informationsverarbeitung. Berlin: VEB Verlag Technik; Heidelberg: Dr. Alfred Hüthig Verlag 1986

Osborne, A.: Einführung in die Mikrocomputertechnik. 4. Aufl. München: te-wi-Verlag 1982

Franke, K.: Einführung in die Mikrorechentechnik. Berlin: VEB Verlag Technik 1984

Blomeyer-Bartenstein: Microcomputer-Technik Z80/Z8/Z8000. München: Hofacker-Verlag 1979

Seifart, M.: Digitale Schaltungen und Schaltkreise. Berlin: VEB Verlag Technik 1982

Hannemann, D.: Einführung in die Microcomputer-Technik: Programmierung, Schaltungstechnik, Anwendung. Essen: Girardent-Verlag 1982

Bennewitz, W.; Podszuweit, H.: Programmierung von Einchipmikrorechnern. REIHE AUTOMATISIERUNGSTECHNIK, Bd. 215. Berlin: VEB Verlag Technik 1985

Kieser, H.; Meder, M.: Mikroprozessortechnik. Aufbau und Anwendung des Mikroprozessorsystems U880. 3. Aufl. Berlin: VEB Verlag Technik 1985

Stuhlmüller, P.: 16-Bit-Generation: Z8000 Aufbau und Anwendung. München: te-wi-Verlag 1980

Feger, O.: Einchip-Microcomputer-Handbuch. Haar bei München: Verlag Markt und Technik 1983

Roth, M.: Mikroprozessoren. Wissenschaftliche Zeitschrift der Technischen Hochschule Ilmenau. Herausgegeben von der KDT-Hochschulsektion. 6. Aufl. 1980

Eckhardt, D.; Konrad, E.; Leupold, W.: Hochintegrierte digitale Schaltungen und ihre Anwendung. 3. Aufl. REIHE AUTOMATISIERUNGSTECHNIK, Bd. 184. Berlin: VEB Verlag Technik 1980

Möschwitzer, A.; Jorke, G.: Mikroelektronische Schaltkreise – Entwurf und Anwendung hochintegrierter Digitalschaltkreise und Mikroprozessoren. 2. Aufl. Berlin: VEB Verlag Technik 1981

Schwarz, W.; Meyer, G.; Eckhardt, D.: Mikrorechner – Wirkungsweise, Programmierung, Applikation. 3. Aufl. Berlin: VEB Verlag Technik 1984

Hoffmann, R.: Rechenwerke und Mikroprogrammierung. REIHE DATENVERARBEITUNG. München–Wien: R. Oldenbourg Verlag 1977

Greenfield, S. E.: The architecture of microcomputers. Cambridge, Massachusetts: Winthrop Publ. Inc. 1980

Kobitzsch, W.: Mikroprozessoren – Aufbau und Wirkungsweise. Teil 1: Grundlagen. REIHE DATENVERARBEITUNG. München–Wien: R. Oldenbourg Verlag 1977

Kobitzsch, W.: Mikroprozessoren – Aufbau und Wirkungsweise. Teil 2: Grundlagen. REIHE DATENVERARBEITUNG. München–Wien: R. Oldenbourg Verlag 1980

Werner, D.: Programmierung von Mikrorechnern – Programmsysteme, Parallele Prozesse, Echtzeitbetriebssysteme. 2. Aufl. Berlin: VEB Verlag Technik 1986; Heidelberg: Dr. Alfred Hüthig Verlag 1984

Lampe, B.; Jorke, G.; Wengel, N.: Algorithmen der Mikrorechentechnik – Maschinenprogrammierung und Interpretertechniken des U880. 2. Aufl. Berlin: VEB Verlag Technik 1984

Claßen, L.: Programmierung des Mikrorechnersystems U880 – K 1520. 5. Aufl. REIHE AUTOMATISIERUNGSTECHNIK, Bd. 192. Berlin: VEB Verlag Technik 1985

Oefler, U; Claßen, L.: Mikroprozessor-Betriebssysteme, Problemorientierte Programmierung U880 (K1520). 2. Aufl. REIHE AUTOMATISIERUNGSTECHNIK, Bd. 201. Berlin: VEB Verlag Technik 1984

Sachwörterverzeichnis

Abstraktion 41
Addierschaltung 84,85
Addier-Subtrahier-Schaltung 85
Addition 26 f.
Adreßcodierung 50
Adresse
–, physikalische 202
–, Segmentanfang 203
–, Segment-Basis 203
–, virtuelle 203
Adressierung
–, blockweise 200
–, direkte 198
–, indirekte 198
–, relative 199
–, unmittelbare 199
Adressierungssystem 50, 58
Akkumulator 144, 161, 167, 226
Algebra, Boolsche 41
Algorithmus 184, 220
ALU (s. Arithmetik-Logik-Einheit)
Analog-Digital-Umsetzer (ADU) 152, 154, 226
Antivalenz (s. EXCLUSIV-ODER)
Äquivalenz 56, 57
Arithmetik-Logik-Einheit (ALU) 85 f., 155, 194, 196
Assemblersprache 185, 186 f., 226
Assemblierung 185, 187, 226
asynchron 33, 210
Ausgabeeinheit 152 f., 154
Ausklammerregel 48, 49

Babbage, Charles 150
Befehle
–, für arithmetische Operationen 164 f.
–, Übertragungs- 161 f.
–, zur Programmorganisation 164
Befehlsliste 160, 161 f.
Bildsignal 219
Bildzeichen (-systeme) 10 f.
Binärzähler 112 f.
Bit 15, 33, 226
–, höchstwertiges (MSB) 33, 227
–, niedrigstwertiges (LSB) 33, 226
Bitmuster 132, 134, 226
Bitmusterfolge 130, 135 f.
Bitmustergenerator 129 f.
Bitmustervergleich 65, 107 f., 151
Buchstabensystem 11 f.
Bündel 35
Bus 63 f., 194, 195 f.
Busanforderung 218
Busbeobachter 107 f.
Busprinzip 36
Byte 33, 226

Cache-Speicher 205
CCD-Anordnung 111 f.
CCITT-Code Nr. 5, 21
Chip 50, 51, 76, 195
–, auswahl (chip select) 79
Codelineal 71
Codierung 9, 66 f., 226
Compiler 186
CRC-Byte 211
Crossassemblierung 188
CTC (counter/timer-circuit) 194, 205 f.

daisy chain 217 f.
Datenwort, Breite des 180
Decodierung 9, 69, 71, 226
Demultiplexer 60 f., 121
Dezimalsystem 13 f.
Digital-Analog-Umsetzer (DAU) 132 f., 152, 153, 226
Digitalrechner (Rechenautomat) 150 f., 153
–, Entwicklung, historische 50
Diodenmatrix 67 f.
Direktoperand 160, 162, 166
Division 15 f., 109, 151
DMA (direkter Speicherzugriff) 212
Doppelwort 80
DRAM (dynamischer RAM) 127
Dualsystem 14 f., 16
Dualzahlengenerator 73, 120 f.

EEPROM 76 f., 127
Eigenleitung 52
Eingabe-Ausgabe-Einheit 194, 209
–, für serielle Datenübertragung 209 f.
–, –, asynchrone 210
–, –, synchrone 211
–, für parallele Datenübertragung 212
Eingabeeinheit 152
Enable (-Signale) 59, 216
ENIAC 51
Entschlüsselung (s. Decodierung)
EPROM 77, 226
Ergibtzeichen 23, 226
EXCLUSIV-ODER 56, 57, 59, 85, 86 f., 151

Flag 158
–, Definitionen 158
–, Zustände 158
Funktionsgenerator, programmierbarer 132 f.

Gray-Code 71

Halbleitermaterial
–, n-leitend 52
–, p-leitend 52
Halbleitertechnik, Einfluß auf Logikschaltungen 50 f.
Handshake-Verfahren 213, 226
Hauptprogramm 138 f., 175
Hexadezimalsystem 16 f.
–, bei Programmniederschrift 185

Information 9 f., 227
–, Austausch zwischen einzelnen Teilsystemen 32 f.
–, Darstellung 10 f.
–, –, mit Hilfe der elektrischen Energie 18
–, Speicherung 19
–, Übertragung
–, –, parallel 11, 35 f., 227
–, –, seriell 11, 21, 33 f., 227
–, Verarbeitung 23 f., 227
–, –, durch einfache Logiksysteme 39 f.
–, –, durch höher entwickelte Logiksysteme 128 f.
Integrationsgrad 51
Interrupt 38, 213, 214 f.
–, maskierbar 216
–, nichtmaskierbarer 216
–, vektorisierter 217
–, –, Prioritätenkette 217 f.
ISO-Code 21
ISR (interrupt service routine) 214

KBit, KByte 79
Kellerspeicher (s. STACK)
Kennlinienfeld
–, Bipolartransistor 53
–, Unipolartransistor 54
Komplementbildung (Zweierkomplement) 28, 84 f.
Kondensator, Ladungsvorgang 91

Leibniz 150
Leitung
–, Eigen- 52
–, Störstellen- 52
LIFO-Prinzip 192
Lochstreifenleser 130
Logik, negative 55 f.
Logikanalysator 108, 189
Logikanordnung, programmierbare 80 f.
Logikpegel, Definitionen 54, 55, 57
Logikschaltungen, elementare 39 f.
Logikschaltungssymbole 56 f.
Logiksystem
–, einfaches, mit Rückführung 88 f.
–, einfaches, ohne Rückführung 39 f.
–, höherentwickeltes 128 f.
Logikzustand, Auswahl auf Leitung 58 f.
LSB 33
LSI 51, 227

Makrobefehlsfolgen, Definition von 192 f.
Makro-/Mikroprogramm-Prinzip 152
Maschinencode 185, 187, 188
Matrixsteuerwerk 143 f., 146, 147, 150, 152, 169 f.
Mehrrechnersystem 89, 214, 219
Meßfühler 152
Meßwertbildung, Anforderung der 181 f.
Meßwerterfassung 177 f., 182
Mittelwertbildung 143 f., 177 f.
Mikroprogramm 151
Mikroprogrammierung 157
Mikroprozessor 150, 155 f., 227
Mikroprozessor, einfacher 158 f.
–, Befehlsliste 159
–, Blockbild 159
–, Definitionstabelle der 16 Befehle 160
–, Entwurfsüberlegungen 165 f.
–, Erläuterung der Befehle 161 f.
–, Funktion des Mikrosteuerwerks 169 f.
–, Mikrosteuerwerk 169
–, Programmablaufplan 168
Mikroprozessor-/Mikrorechner-Konzept, Weiterentwicklung 194 f.
Mikrorechner 143, 150, 155, 227
Mikrorechner, Anwendungsmöglichkeiten 220 f.
–, Abtastoszillograf 224
–, chemische Verfahrenstechnik 221
–, Diagnoseverfahren 225
–, Fernsprechvermittlungstechnik 32, 223
–, Funksprechanlagen, mobile 223
–, Funktionsgeneratoren 224
–, Heimelektronik 225
–, Kraftfahrzeug 222
–, Meßgeräte 223
–, Steuerung technischer Prozesse 220
–, Treibhäuser 222
–, Überwachung von Gefahrenherden 222
–, Weltraumflug, unbemannter 222
Mikrorechner-Entwicklungssystem 187, 188
Mikrosteuerwerk 155, 157, 165 f., 204
Mittelwert, arithmetischer 143, 178 f., 183
Mnemonik 160 f.
Morsealphabet 21
Morsetelegrafie 9
MSB 33, 227
MSI 51, 227
Multiplexer 60 f., 86 f., 206, 207
Multiplikation 15, 16, 151
Multivibrator
–, astabiler 91 f.
–, monostabiler 89 f.
Musterelemente 135 f.
Mustergenerator 135 f.
–, Verbesserungsmöglichkeiten des 141 f.

NAND-Funktion 44 f.
NAND-Gatter
–, TTL 53
–, MOS 55

NAND-Glied 56
Negation 43 f.
Negator 56
–, MOS-Technik 54
Neumann, von 150
Nibble 80
NOR-Funktion 45, 56
NOR-Gatter, MOS-Technik 55
NOR, verdrahtetes (wired NOR) 64 f.
NVRAM 127

ODER-Funktion 42 f., 56
ODER-Glied 56
Oktalsystem 16 f.

Parallel-Serien-Wandler 109, 210
Paritätsbit 33, 210
PIO 212
Pipelinig-Verfahren 204 f.
PLA (programmable logic array) 80 f., 146, 227
Polling 213
Programmablaufplan 23 f.
–, gestreckter 23
–, Meßwertaufnahme mit Mittelwertbildung 145, 179
–, Mikroprozessor, allgemein 156
–, Mikroprozessor, einfacher 168
–, mikrorechnergesteuertes Digitalvoltmeter 223
–, monostabiler Multivibrator 90
–, Mustererzeugung 139, 142
–, Sortierung 31
–, zyklischer 23 f.
Programm, Haupt- 138, 175
Programm, Mikro- 152, 157
Programm, Unter- 138, 189 f., 192
Programme, die ersten kleinen 173 f.
Programmherstellung 184
–, Unterlagen für Programmierer 184
–, in Assemblersprache 185, 186 f.
Programmierbeispiel 177 f.
Programmieren
–, eines ROM 76 f.
–, von Baugruppen 86
Programmierung, Mikro- 157
Programmniederschrift 185 f.
Programmspeicher, Mikro- 151
PROM 76, 227
Prozeßkontrolleinheit 152, 153, 154
Pulsgenerator 91

Quellenstreifen 187

RAM 123 f., 130, 194, 227
RAM-Zelle
–, bipolar 123 f.
–, MOS 124 f.
Register 33 f., 63, 104 f., 227
–, Auffang- 159, 162, 163, 166, 167
–, Befehls- 155, 157, 159, 165, 172, 227
–, Flag- 155, 157, 159, 165, 227
–, zur parallelen Dateneingabe 105
–, zur seriellen Dateneingabe 105 f., 212
Registerblock 196 f.
Ringzähler 116
ROM 72 f., 131, 134
Rückführung 34, 88, 89
Rücksetzen 92, 102, 117
Rücksetzlage 93 f.

SAM 212
Sammelschienenübertragung (s. Bus)
Schaltalgebra 41 f.
–, Regeln und Anwendungen 46 f.
Schaltbelegungstabelle 41 f.
Schalter, prellfreier 94
Schaltkontakt 39 f.
–, Zustandsvariable 40
Schaltung, kombinatorisch 39
Schieberegister 33 f., 105 f., 212, 227
–, als Verzögerungsglied 107
–, als Serien-Parallel-Wandler 109
–, Anwendungsbeispiele 109 f.
–, für serielle oder parallele Eingabe 105, 108
–, mit taktflankengesteuerten D-Triggern 107
–, MOS-Technik 110 f.
Segment 201
Seite 201
Semikunden-IC 155
Serien-Parallel-Wandlung 109
Setzen 92, 102
Setzlage 93 f.
Signal 10, 227
Signaleinschalter (Torschaltung) 59 f.
Signalflußbild 57
Signalumschalter 60 f.
SIO 211
Sortierung
–, nach Stromverstärkungswerten 31 f.
–, nach Toleranzbereichen 70 f.
Spannungsstufengenerator 132 f.
Speicherdichte 19
Speicher
–, externer 156
–, holografischer 19, 226
–, Kapazität 79, 156
–, Keller- (s. STACK)
–, Notizbuch- 155, 157
–, Organisation 77 f.
–, Register- (s. Register)
Speicherverwaltungseinheit 194, 201
Spike 115, 227
Sprachsignal 219
Sprachkommunikation 89, 219
Sprung 25, 160, 163, 177, 227
SRAM (statischer RAM) 126
SSI 51, 227
STACK 156, 158, 163, 164, 192, 227
STACK-Pointer 191, 192
Stelleinrichtungen 154
Stellglieder 154
Steuersystem, universelles 150 f.
Steuerung
–, technologischer Prozeß 220
–, Waschmaschine 73 f.
Steuerungssystem, automatisches 152 f., 154
Steuerwerk
–, Matrix- 143 f., 146, 147, 150, 152, 169 f.
–, Mikro- 157, 169 f., 204 f.
Subtraktion 28 f.
Symbol 10
synchron 33, 211
Synchronbyte 211
System, informationsverarbeitendes (I-System) 37
–, gleichberechtigtes 37 f.
–, hierarchisches 37

Taktgenerator 73, 144, 155
Taktimpuls 97 f.
Theoreme von De Morgan 45 f., 48, 82
Time-sharing 60
Torschaltung 59 f., 149, 172
Transformation der Informationsdarstellung 19 f.
Transistor
–, als Schalter 51 f.
–, bipolarer 52 f.
–, MOS 54 f.
–, unipolarer 54 f.
–, Verzögerungszeit 51, 53
Transputer 219
Trap 218
Tigger 92 f., 228
–, D- 97
–, –, mit Taktflankensteuerung 97 f.
–, JK-Master-Slave 100 f.
–, RS- 92 f.
–, –, getakteter 96 f.
–, –, mit vorgeschalteten Negatoren 95 f.
–, Zusammenschaltung von 102 f.
Tristate-Gatter 64, 79, 228

UART 211
Überlauf 28, 31
Übertrag 27 f., 83 f., 228
Umcodierung 19 f., 66 f., 228
UND-Funktion 39 f.
Ungültige Ergebnisse, Erkennung 30 f., 84 f.
Unterbrechungssystem, Gestaltung des 214
Unterprogramm 138, 189 f., 192
Unterprogrammtechnik 189 f.
USART 211

Variable, zur Kennzeichnung von Zuständen 40
Vergleicher 65
Verschlüsselung (s. Codierung)
Vierfachwort 80
VLSI 51, 228
Volladder 83 f.
Vollkunden-IC 155
Vorzeichenbit 26 f.

Wartesignal (WAIT) 155, 157, 159, 168, 169, 181 f.
Wartezustand 149
Wilkes, M. V. 146, 150, 152
Wirtsrechner 188, 228

Zahlensysteme 12 f.
Zahlentransformation 21 f.
Zähler, binär 112 f.
–, als Element für zyklischen elektronischen Schalter 121
–, als Verzögerungseinrichtung 121 f.
–, asynchron 112 f.
–, zur Uhrzeitbildung 122
–, –, rückwärtszählend 115 f.
–, –, vorwärtszählend 112 f.
–, –, vor-/rückwärtszählend 116 f.
–, synchron 117 f.
–, –, vorwärtszählend 117
–, –, vor-/rückwärtszählend 117 f.
Zählerbaustein, universeller 120
Zähler, Programm- 155, 156, 159, 163, 164, 197
Zahlwortdefinition 26
Zeichen 10, 228
Zeile 201
Zeit-Multiplex-Verfahren 60
Zuordner 66 f., 228
–, Anwendungsbeispiel 70 f.
–, universeller 71 f.
Zuse 50
Zustand 17 f., 228
Zustandserkennung, automatische 31 f.
Zustandssignale 155, 157, 173 f.
Zustandssystem 17 f.
Zweierkomplement 23 f., 228

Aus Rezensionen:

„Das vorliegende Buch verfolgt das Ziel, einen möglichst umfassenden Überblick über die Wirkungsweise und die Einsatzmöglichkeiten digitaler Schaltkreise zu vermitteln. In der Fülle der Veröffentlichungen zur funktionellen und applikativen Seite der Mikroelektronik nimmt es durch seine bewußt einfache und didaktisch geschickte Gestaltung einen besonderen Platz ein.

Der Autor ist bestrebt, vorrangig dem lernenden Leser durch deutliches Herausarbeiten von Grundprinzipien und durch gut gewählte Demonstrationsbeispiele einen Leitfaden durch das auf den ersten Blick so unübersichtliche und komplizierte Gebiet der Digitaltechnik in die Hand zu geben." –rfe–

„Auch im Vergleich mit entsprechender internationaler Einführungsliteratur kann die Publikation als sehr sorgfältig und gelungen angesehen werden. Nicht allein der angesprochene Leserkreis wird das Buch zu Studienzwecken mit großem Gewinn nutzen, auch der Fachmann wird viele methodische Anregungen für die Wissensvermittlung entnehmen können." –msr–

ISBN 3-341-00069-0